X1＋turbo マシン語読本

パソコンテレビ

清水保弘著

工学図書株式会社

X1 + turbo マシン語読本

パソコンテレビ

清水保弘著

工学図書株式会社

ま え が き

マシン語の魅力って何でしょう？　初期のマイコン・ブームの頃に，マイコン（ワンボード・キットでした）に取り組んだ方々が書かれた文章を読むと，当時はマイコンを動かすのにはマシン語しかなく，小さなBASICを愛機で走らせることがホビィストの憧れであったようです。

しかし，パソコンがこれほど普及し，BASICが動くのが当然のようになっている現在，最初の問はもう少し違った響きを持っているように思えます。言うならば，「私たちの愛機を動かしているのは，実はマシン語なのだ！」と改めて確認するときの驚きというか，感動というか…。

本書は，私たちの愛機であるX1シリーズおよびturboシリーズをマシン語で操ってみたいと思い立った皆さんへ贈るガイドブックです。

X1およびturboには，Z80AというLSI（大規模集積回路）が塔載されていて，私たちの愛機をマシン語で操るのは，Z80Aにマシン語の命令を実行させることにほかなりません。本書では，Z80Aの命令を整理して説明してありますから，BASICの命令を少しずつ覚えていったように，Z80Aのマシン語をマスターできるように工夫されています。

また，皆さんの愛機を実際に動かしながら学習できるよう豊富な「実例」と「実験材料」を用意し，全体として「読み物」風に書かれています。題名を「マシン語読本」としたのもそのためです。

最近では，X1シリーズ，X1 turboシリーズのハードウェアの詳細を扱った本が出はじめました。本書は，読者の皆さんがこれらの本へ進むための「基礎編」として執筆されました。本書でマシン語の基礎をマスターされた皆さんの前には，さらにドラマチックな旅が待っていることでしょう。

1985年夏

清 水 保 弘

目　　次

第1章　マシン語を始めよう

第2章　Z80 CPUとハンド・アセンブル

第3章　データの転送

第4章　マシン語での計算

第5章　プログラムの流れをかえる

第6章　入出力と画面制御

第7章　桁ずらし，回転，ビット操作

第8章　交換命令

第9章　ブロック命令

第10章　CPU制御命令

第11章　BASICとマシン語の共存

第1章　マシン語を始めよう

これから読者の皆さんとともに，X1シリーズ(turboシリーズも含めて)のマシン語の基礎をマスターする旅に出発いたします。途中の眺めを楽しみながら，ノンビリおつきあいください。

皆さんの多くは，ふだんBASICを用いて愛機と「対話」されていることでしょう。そんなとき，ふと「パソコンは片言の英語がわかるのでは？」という錯覚に陥ることはありませんか？　マシン語の勉強を始めるにあたって，まず大切なことは「コンピュータにとって，本来わかる言葉とは何か？」についての正しい認識を持つことであると思います。そこで，話は少し大袈裟になりますが，コンピュータはどのような仕組みで動作するのかについての知識を簡単に復習することからスタートしましょう。

1.1　コンピュータの動作原理

X1シリーズのようなパソコンにしても，電卓にしても，機械に組み込まれた制御用マイコンにしても，はたまた汎用の大型コンピュータ(メインフレーム)にしても，1つのまとまった働きをしているコンピュータは，すべて次の4つの部分からなっています(図1.1)。

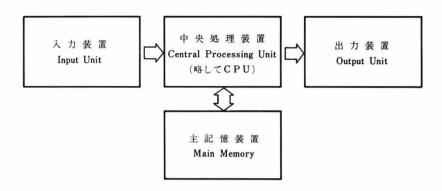
未来のコンピュータは別にして，現在までのコンピュータの動作原理は，フォン・ノイマンという偉い数学者が提案した「プログラム内蔵型逐次処理」という方式が採り入れられています。大昔の計算機は，各々の仕事内容ごとに機械が作られていたのですが，フォン・ノイマン先生の提案は，計算機の本体(中央処理装置＝CPU)は極めて汎用性の高いものにしておき，主記憶装置に蓄積されたデータを命令の列(プログラム)と解釈して仕事をするようにさせよう！というものでした。こうすれば，プログラムを変えるだけで，同じ機械を別の

○入力装置には，キーボード，マウス，ジョイスティックなどがある．
○出力装置には，ディスプレイ，プリンタなどがある．
○入力装置と出力装置をまとめて，**入出力装置（Input Output Unit）**あるいは**I/O**とよぶことがある．
○主記憶装置は，コンピュータ内部に設けられていて，**内部記憶装置**ともよばれる．これ以外に，フロッピー・ディスク装置やカセット・データ・レコーダなどが**補助記憶装置**（あるいは**外部記憶装置**）として用いられる．

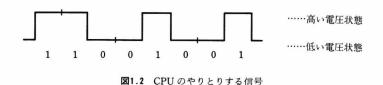
図1.1 コンピュータの構成

用途に使えるわけですね。X1シリーズも当然その系譜を受け継いでいます。プログラム1つで，X1がゲーム・マシンになったり，あるいはワープロに変身したりするのはそのためなのです。

最近の半導体技術の飛躍的な発展は，CPUを手のひらにのるほど小さい**LSI（大規模集積回路）**として実現させました。いわゆる**マイクロ・プロセッサ（マイコン）**の誕生です。マイコンCPUにおいては，CPUがやりとりする信号は，ある微小基準電圧より電圧が高いか低いかで組み立てられる電気信号です。これを表わすのに，ふつう高い電圧の状態を1で，低い方を0で表わし，電気信号を1と0の数字列――**2進数**――で表記します。

2進数では，各桁を**ビット（bit）**とよび，*n*桁の2進数は***n*****ビット**であるといいます。そこで，CPUの処理する情報を測る最小単位としてビットが用いられます。

CPUのやりとりする信号

図1.2 CPUのやりとりする信号

X1シリーズに搭載されているCPUは，米ザイログ社が開発したZ80Aという型番をもつLSIです。Z80Aには，一度に8ビットの情報を処理する能力があります。Z80Aが**8ビットCPU**とよばれ，またそれを用いているX1シリーズが「**8ビットマシン**」とよばれる理由はこういうわけです。

さて，8ビットの情報は1まとまりとして扱われることが多く，そのときには，**バイト（byte）**という単位が使われます。

〔例〕 1バイト＝8ビット
　　　 2バイト＝16ビット

X1シリーズのような8ビットマシンの主記憶装置は，1バイトずつに分けられた記憶場所が整然と並べられていて，各記憶場所は**番地（アドレス）**とよばれる数字により区別されます。番地は，0番地から始まって，1番地，2番地，…というように整数値でつけられています。Z80AというCPUでは，番地の数字は最大65535番地までつけることができます。各番地には1バイトの情報が記憶され，番地は0番地から65535番地まで合計65536個ありますから，Z80Aの主記憶装置の記憶容量は最大65536バイトであると言うことができます。このような大きな数字を扱うのに，コンピュータでは$1024=2^{10}$をK（ケーと読む）という単位で表わします（コンピュータでは2進数を用いる関係で，2の何乗という数が都合よく，この中で最も通常の1000に近い数字が$2^{10}=1024$というわけです）。$65536=64\times1024$ですから，「Z80Aが直接にやりとりできる

主記憶装置の最大容量は64Kバイトである」と言えます。X1シリーズにおいては，64Kバイト分の主記憶装置が本体に内蔵されています。

1.2　メモリーをのぞく

では，X1シリーズの主記憶装置内にどのように情報が記憶されているか，実際に簡単なBASICプログラムを使って見てみることにしましょう。主記憶装置は英語で main memory（**メイン・メモリー**）とよばれますが，これから頻繁に登場しますので，以後，単に**メモリー**と略称します。

BASICを用いて，メモリーを「のぞく」(peek)には，そのものズバリ **PEEK** という関数を使えばよいのです。たとえば，

M＝PEEK (A)

とすると変数 **A** の示す番地のメモリー内容である1バイトが変数 **M** に（10進数値として）代入されます。**PEEK** 関数を利用した次の短いプログラムをご覧ください。

```
100 REM  ﾒﾓﾘｰ ｦ ﾉｿﾞｸ
110 '
120 INIT : WIDTH 40 : CONSOLE 0,24
130 '
140 INPUT "ﾒﾓﾘｰ ﾉ ﾅﾝﾊﾞﾝﾁ ｶﾗ ﾋｮｳｼﾞ ｼﾏｽｶ ? ",N$
150 IF INSTR("Eeｲ",N$)>0 THEN 300
160 N=VAL(N$)
170 LOCATE 13,CSRLIN : PRINT "ﾒﾓﾘｰ ﾅｲﾖｳ" : PRINT
180 FOR J=N TO N+19
190   A=J : IF A>65535! THEN A=A-65536!
200   IF A<0 THEN A=A+65536!
210   PRINT USING "##### ﾊﾞﾝﾁ : ";A;
220   M=PEEK(A)
230   PRINT RIGHT$("0000000"+BIN$(M),8);
240   PRINT USING " = ### ( 10ｼﾝ )";M
250 NEXT J
260 N$=STR$(J)
270 PRINT
280 GOTO 140
290 '
300 INIT : END
```

図1.3

プログラムを打ち込み，RUN すると，メモリー内容を表示する開始番地を尋ねてくるので，0から65535までの好きな番地を入力してください。結果は図1.4のように表示されます（表示内容が図1.4の通りでなくてもアワテナイでください。X1とturboでも違うでしょうし，また走らせている状態によっても異なりますから）。

```
ﾒﾓﾘｰ ﾉ ﾅﾝﾊﾞﾝﾁ ｶﾗ ﾋｮｳｼﾞ ｼﾏｽｶ ? 0
                 ﾒﾓﾘｰ ﾅｲﾖｳ

    0 ﾊﾞﾝﾁ : 11000011 = 195 ( 10ｼﾝ )
    1 ﾊﾞﾝﾁ : 11111010 = 250 ( 10ｼﾝ )
    2 ﾊﾞﾝﾁ : 00000000 =   0 ( 10ｼﾝ )
    3 ﾊﾞﾝﾁ : 11000011 = 195 ( 10ｼﾝ )
    4 ﾊﾞﾝﾁ : 01111100 = 124 ( 10ｼﾝ )
    5 ﾊﾞﾝﾁ : 00000001 =   1 ( 10ｼﾝ )
    6 ﾊﾞﾝﾁ : 11111111 = 255 ( 10ｼﾝ )
    7 ﾊﾞﾝﾁ : 00101000 =  40 ( 10ｼﾝ )
    8 ﾊﾞﾝﾁ : 11000011 = 195 ( 10ｼﾝ )
    9 ﾊﾞﾝﾁ : 11010011 = 211 ( 10ｼﾝ )
   10 ﾊﾞﾝﾁ : 00000011 =   3 ( 10ｼﾝ )
   11 ﾊﾞﾝﾁ : 11000011 = 195 ( 10ｼﾝ )
   12 ﾊﾞﾝﾁ : 10000011 = 131 ( 10ｼﾝ )
   13 ﾊﾞﾝﾁ : 00000100 =   4 ( 10ｼﾝ )
   14 ﾊﾞﾝﾁ : 00001101 =  13 ( 10ｼﾝ )
   15 ﾊﾞﾝﾁ : 00010010 =  18 ( 10ｼﾝ )
   16 ﾊﾞﾝﾁ : 11000011 = 195 ( 10ｼﾝ )
   17 ﾊﾞﾝﾁ : 11010011 = 211 ( 10ｼﾝ )
   18 ﾊﾞﾝﾁ : 00000011 =   3 ( 10ｼﾝ )
   19 ﾊﾞﾝﾁ : 11000011 = 195 ( 10ｼﾝ )
```

```
ﾒﾓﾘｰ ﾉ ﾅﾝﾊﾞﾝﾁ ｶﾗ ﾋｮｳｼﾞ ｼﾏｽｶ ? 65530
                 ﾒﾓﾘｰ ﾅｲﾖｳ

65530 ﾊﾞﾝﾁ : 00000011 =   3 ( 10ｼﾝ )
65531 ﾊﾞﾝﾁ : 00010001 =  17 ( 10ｼﾝ )
65532 ﾊﾞﾝﾁ : 11110000 = 240 ( 10ｼﾝ )
65533 ﾊﾞﾝﾁ : 00010000 =  16 ( 10ｼﾝ )
65534 ﾊﾞﾝﾁ : 00000011 =   3 ( 10ｼﾝ )
65535 ﾊﾞﾝﾁ : 00010000 =  16 ( 10ｼﾝ )
    0 ﾊﾞﾝﾁ : 11000011 = 195 ( 10ｼﾝ )
    1 ﾊﾞﾝﾁ : 11111010 = 250 ( 10ｼﾝ )
    2 ﾊﾞﾝﾁ : 00000000 =   0 ( 10ｼﾝ )
    3 ﾊﾞﾝﾁ : 11000011 = 195 ( 10ｼﾝ )
    4 ﾊﾞﾝﾁ : 01111100 = 124 ( 10ｼﾝ )
    5 ﾊﾞﾝﾁ : 00000001 =   1 ( 10ｼﾝ )
    6 ﾊﾞﾝﾁ : 11111111 = 255 ( 10ｼﾝ )
    7 ﾊﾞﾝﾁ : 00101000 =  40 ( 10ｼﾝ )
    8 ﾊﾞﾝﾁ : 11000011 = 195 ( 10ｼﾝ )
    9 ﾊﾞﾝﾁ : 11010011 = 211 ( 10ｼﾝ )
   10 ﾊﾞﾝﾁ : 00000011 =   3 ( 10ｼﾝ )
   11 ﾊﾞﾝﾁ : 11000011 = 195 ( 10ｼﾝ )
   12 ﾊﾞﾝﾁ : 10000011 = 131 ( 10ｼﾝ )
   13 ﾊﾞﾝﾁ : 00000100 =   4 ( 10ｼﾝ )
```

図1.4 実行例

中央の列が，各番地の1バイト分を2進表示したもの，右の列が対応する10進表示を意味しています。開始番地を尋ねられているときに，単にリターンキーを押すと，次の20バイトが表示されます。また，Eを入力するとプログラムが終了します。いろいろな番地を入力して，メモリー内の情報の姿を観察しましょう。

実は，0番地から始まる部分には，私たちがお世話になっているBASIC言語を解釈するためのシステム・プログラム（BASICインタプリタ）が置かれているのです。これがプログラムの「真の姿」なのです。読者の皆さんは「何と奇怪な!?」と感じられるでしょうか，それとも「何て美しい！」と感じられるでしょうか。

1.3　16進表記法

前節のBASICプログラムを走らせて遊んでいると，1つ気づくことがあります。2進表示は確かに情報が記憶されている様子を忠実に反映しているのですが，何とも「非人間的」であり私たちにはわかりにくい（昔のコンピュータでは，このようなパターンを紙テープに穴をあけて使っていました。かつてのSF映画でそうした紙テープを眺めてフムフムと理解している天才科学者が登場しましたが，そのような天才は別格として！）。かといって，私たちがなじんでいる10進表記では，実際の姿とかけ離れている。これらの中間はないものだろうか？　と考えたくなるわけです。

こんなウマイことをある程度実現してくれるのが，**16進数**を用いた表記法です。16進数では16種類の数字が用いられます。0から9までの10個は10進数と共通なのですが，これ以外にアルファベットのAからFの6個を使う習慣になっています。1桁の16進数と10進数との対応は次の通りです。

16進数	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10進数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

図1.5　16進数と10進数

なぜ16進数を使うと都合よいかの秘密は，$2^4=16$という関係にあります。すなわち，16進数の１桁が，２進数での４ビットに対応し，16進数２桁で１バイトの情報を表現できるからです。１桁の16進数と４ビットの２進数との対応を以下に掲げます。

16進数	２進数（４ビット）
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

図1.6　16進数と２進数

こうして，２進数の持つ構造は忠実に16進数に反映されますから，16進表記から２進表示を復元することは容易です。また，16進表記では桁数が少なくて済みますし，私たちにとってもある程度，数としてイメージを浮かべることが可能です。

２進数を16進数に直すには，右から４ビットずつ区切り「対応表」(図1.6)

のように変換していけばよいのです。

[例]

1100　1001　→　16進数　C9

C　9

16進表記では10進数と共通の数字が使われますから，16進数であることを明示するために本書では，16進数の後にHをつけて表示します(Hは16進数を意味する英語 hexadecimal number の頭文字)。このほかに HuBASIC のプログラム中では，16進数の頭に &H をつけて使われます。また，頭に $ をつける流儀や，数字の後に小さく $_{(16)}$ と添字をつけて表わす方法などもあります。

[16進数を明示する方法]

C9H　(→本書で用いる)

&HC9　(→ HuBASIC で用いる)

$C9, C9$_{(16)}$　など。

前節のプログラムを一部改良して，番地の数字と，メモリー内容の表示に16進表記をとり入れたものが以下のリストです。

```
100 REM  ﾒﾓﾘｰ ｦ ﾉｿﾞｸ  ( 16 ｼﾝ ﾋｮｳｼﾞ )
110 '
120 INIT : WIDTH 40 : CONSOLE 0,24
130 '
140 INPUT "ﾒﾓﾘｰ ﾉ ﾅﾝﾊﾞﾝﾁ ｶﾗ ﾋｮｳｼﾞ ｼﾏｽｶ ? ",N$
150 IF INSTR("Eeｲ",N$)>0 THEN 300
160 N=VAL(N$)
170 LOCATE 13,CSRLIN : PRINT "ﾒﾓﾘｰ ﾅｲﾖｳ" : PRINT
180 FOR J=N TO N+19
190   A=J : IF A>65535! THEN A=A-65536!
200    IF A<0 THEN A=A+65536!
210   PRINT USING "##### ";A;
215   PRINT "( ";RIGHT$("000"+HEX$(A),4);"H ) ﾊﾞﾝﾁ : ";
220   M=PEEK(A)
230   PRINT RIGHT$("0000000"+BIN$(M),8);
240   PRINT " = ";RIGHT$("0"+HEX$(M),2);"H "
250 NEXT J
260 N$=STR$(J)
270 PRINT
280 GOTO 140
290 '
300 INIT : END
```

図1.7

今度は RUN すると以下のような画面表示になるはずです。

```
ﾒﾓﾘｰ ﾉ ﾅﾝﾊﾞﾝﾁ ｶﾗ ﾋｮｳｼﾞ ｼﾏｽｶ ? 0
              ﾒﾓﾘｰ ﾅｲﾖｳ

    0 ( 0000H ) ﾊﾞﾝﾁ : 11000011 = C3H
    1 ( 0001H ) ﾊﾞﾝﾁ : 11111010 = FAH
    2 ( 0002H ) ﾊﾞﾝﾁ : 00000000 = 00H
    3 ( 0003H ) ﾊﾞﾝﾁ : 11000011 = C3H
    4 ( 0004H ) ﾊﾞﾝﾁ : 01111100 = 7CH
    5 ( 0005H ) ﾊﾞﾝﾁ : 00000001 = 01H
    6 ( 0006H ) ﾊﾞﾝﾁ : 11111111 = FFH
    7 ( 0007H ) ﾊﾞﾝﾁ : 00101000 = 28H
    8 ( 0008H ) ﾊﾞﾝﾁ : 11000011 = C3H
    9 ( 0009H ) ﾊﾞﾝﾁ : 11010011 = D3H
   10 ( 000AH ) ﾊﾞﾝﾁ : 00000011 = 03H
   11 ( 000BH ) ﾊﾞﾝﾁ : 11000011 = C3H
   12 ( 000CH ) ﾊﾞﾝﾁ : 10000011 = 83H
   13 ( 000DH ) ﾊﾞﾝﾁ : 00000100 = 04H
   14 ( 000EH ) ﾊﾞﾝﾁ : 00010111 = 17H
   15 ( 000FH ) ﾊﾞﾝﾁ : 00010010 = 12H
   16 ( 0010H ) ﾊﾞﾝﾁ : 11000011 = C3H
   17 ( 0011H ) ﾊﾞﾝﾁ : 11010011 = D3H
   18 ( 0012H ) ﾊﾞﾝﾁ : 00000011 = 03H
   19 ( 0013H ) ﾊﾞﾝﾁ : 11000011 = C3H
```

```
ﾒﾓﾘｰ ﾉ ﾅﾝﾊﾞﾝﾁ ｶﾗ ﾋｮｳｼﾞ ｼﾏｽｶ ? 65530
              ﾒﾓﾘｰ ﾅｲﾖｳ

65530 ( FFFAH ) ﾊﾞﾝﾁ : 00000000 = 00H
65531 ( FFFBH ) ﾊﾞﾝﾁ : 00000000 = 00H
65532 ( FFFCH ) ﾊﾞﾝﾁ : 00000000 = 00H
65533 ( FFFDH ) ﾊﾞﾝﾁ : 00000000 = 00H
65534 ( FFFEH ) ﾊﾞﾝﾁ : 00000000 = 00H
65535 ( FFFFH ) ﾊﾞﾝﾁ : 00000000 = 00H
    0 ( 0000H ) ﾊﾞﾝﾁ : 11000011 = C3H
    1 ( 0001H ) ﾊﾞﾝﾁ : 11111010 = FAH
    2 ( 0002H ) ﾊﾞﾝﾁ : 00000000 = 00H
    3 ( 0003H ) ﾊﾞﾝﾁ : 11000011 = C3H
    4 ( 0004H ) ﾊﾞﾝﾁ : 01111100 = 7CH
    5 ( 0005H ) ﾊﾞﾝﾁ : 00000001 = 01H
    6 ( 0006H ) ﾊﾞﾝﾁ : 11111111 = FFH
    7 ( 0007H ) ﾊﾞﾝﾁ : 00101000 = 28H
    8 ( 0008H ) ﾊﾞﾝﾁ : 11000011 = C3H
    9 ( 0009H ) ﾊﾞﾝﾁ : 11010011 = D3H
   10 ( 000AH ) ﾊﾞﾝﾁ : 00000011 = 03H
   11 ( 000BH ) ﾊﾞﾝﾁ : 11000011 = C3H
   12 ( 000CH ) ﾊﾞﾝﾁ : 10000011 = 83H
   13 ( 000DH ) ﾊﾞﾝﾁ : 00000100 = 04H
```

図1.8　実行例

メモリーの各番地の内容(1バイト)は2桁の16進数で表わされます。また，もう1つ注目していただきたいのは，番地を表わす数字自体も16進表記していることです。最大の番地である65535はFFFFHという4桁の16進数で表わされますから，以後，メモリーの番地自身を示すのに4桁の16進数が使われます。4桁に満たないときは，0E9CH番地というように上位桁に0を補っておくのが

普通のようです。

こうして私たちは，X1シリーズのマシン語を学ぶための最初の大きな壁（メモリー内の情報を表記する方法）を越えたことになります。次節からは，いよいよ正式に「マシン語」が登場します。

1.4 マシン語と CPU

前節までで，私たちはX1シリーズのメモリーの内容を観察してみました。それは私たちにとっては極めて「無味乾燥」な（16進表記された）1バイト情報の列でした。ところが，これこそX1のCPUであるZ80Aの唯一理解できる「言葉」なのです。CPUはこのような情報の列を（ふつう何バイトかずつまとめて）「命令」として解釈し，それに応じた動作をするように設計されています。これをCPUの**マシン語（機械語）**とよびます。

たとえば，x番地から$x+2$番地までのメモリー内容が右のようになっていたとしましょう。Z80Aは，x番地からの情報を3バイト命令「00FAH番地へジャンプせよ。」として解釈するように設計されています。これはBASICの**GOTO**文に相当するマシン語命令の例です。

メモリー内の情報の列をどのような「命令」として解釈するかは，各々のCPUの設計方針によって異なっています。パソコンに使われる8ビットCPUでは，図1.9の型番でよばれるLSIたちが有名です。

有名な8ビットCPUは表のように，80系，68系とよばれる2系統に大別されます。同系のCPUどうしでは，少しの違いはあるもののマシン語の体系はかなり似ていますが，80系と68系とではマシン語は全く異なっています。私たちのX1シリーズ（turboを含めて）では，CPUはZ80の高速版の1つであるZ80Aが使われています（Z80とZ80Aとは，CPUを働かせるクロック周波数が異なるだけで，マシン語は全く同じです）。ですから，私たちの学習目標はZ80のマ

シン語ということになるわけですね。

系統	CPU型番	搭載している主なパソコン
80系	インテル 8080	(かつてのワンボード・マイコン)
	ザイログ Z80 (注)	シャープ X1シリーズ X1 turboシリーズ MZシリーズ NEC PC-8000シリーズ PC-8800シリーズ
68系	モトローラ 6809	富士通 FM-7シリーズ 日立 S1
	モステク 6502	アップルII

(注) Z80AはZ80の「高速版」の1つ.

図1.9 8ビットCPU

1.5 モニター

私たちはこれから，メモリー内容を読みとったり，メモリーにマシン語プログラムやデータを書き込んだりしながら，マシン語の勉強をしていくことになりますが，そのために不可欠な道具の使用法をマスターしておきましょう。

X1シリーズのHuBASICでは，ユーザーがマシン語プログラムを利用しやすいように，**マシン語モニター（機械語モニター）**とよばれるシステムを内蔵しています。今後，この言葉もたくさん登場しますから，単に**モニター**とよぶことにしましょう。モニター（monitor）の原義は「監視する，制御する」で，今の場合，ユーザーによるマシン語の利用を監視・制御するシステムというほどの意味あいです。

さて，BASICからモニターをよび出すには**MON**というBASICコマンドを使います。**MON**とキー入力してリターンキーを押すと図のように＊(アスタリスク）が表示され，カーソル点滅となります。

```
Ok
MON
*■
```

図1.10

これがモニターの管理下に入った印です。BASICのときには，**OK**表示でカーソルが点滅しました。この状態は，BASICのコマンドを受けつける状態で**「BASICのコマンドレベル」**とよばれます。それに対し，＊表示でカーソル点滅時には，モニター用の特別なコマンドしか受けつけません。これを**「モニターのコマンドレベル」**とよびます。

モニターのコマンドレベルからBASICへ復帰するには，単に**R**と入力してリターンキーを押してください。

図1.11

OKが表示され，BASICのコマンドレベルに戻れましたね。こうして私たちは今後の学習に欠かせないこと ―― BASICとモニター間の往復 ―― をマスターしました。

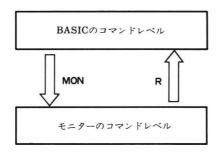



図1.12

さて，もういつでも好きなときにBASICへ戻れますから，安心してMONにより，モニターへ入ってください。モニターのコマンドレベルで用いることのできる**モニター・コマンド**は，原則としてアルファベット1文字の形をしています。X1シリーズとturboシリーズで共通なモニター・コマンドは，次の10種類です。

コマンド	名　　称
D	ダンプ
M	メモリーセット
G	ゴーサブ
T	トランスファ
F	ファインド
S	セーブ
L	ロード
V	ベリファイ
P	プリンタスイッチ
R	リターン

（注）turbo BASIC（SHARP HuBASIC CZ-8FB02）の内蔵モニターでは，機能がさらに拡張されていますが，これらについてはturboのマニュアルを参照してください．

図1.13　モニター・コマンド

モニター・コマンドを自由に使いこなせるようにすることは，マシン語学習に不可欠ですから，次節より一通り解説いたします。是非マスターしてください。

1.6　メモリーのダンプ（Dコマンド）

まず最初に覚えるのは，Dコマンドの使い方です。このコマンドは，「どさりと降ろす」を意味するdump（ダンプ）の頭文字をとったものです。「ダンプカ

ー」のダンプです。コンピュータ用語としての**「ダンプ」**は，メモリー内に記憶されている情報を出力装置へ出力することを指します。

Dコマンドは，次の3通りのいずれかの方法で用います。

書　　式	機　　能	例
＊D　先頭アドレス　最終アドレス ↑ 空白1文字分 を入れること	指定された先頭アドレスより，最終アドレスまでを，ダンプする．	＊D　10F0　1FFF
＊D　先頭アドレス	指定された先頭アドレスより，128バイト分を，ダンプする．（注）	＊D　2000
＊D	現在，システムが記憶しているアドレスより，128バイト分をダンプする．（注）	＊D

（注）　turboのモニターでは，画面が40桁モードのとき128バイト分，80桁モードのとき256バイト分をダンプします．

図1.14　Dコマンドの書式

メモリーのアドレス（番地）は4桁以内の16進数で指定します。その際，16進を意味するHはつけなくても構いません。

次に実行例を挙げます。turboシリーズの方は，あらかじめ**WIDTH 40：KMODE 0**を実行しておいてください。下に掲げた例はX1のディスクBASIC（CZ-8FB01）のものです。テープBASIC（CZ-8CB01），turbo BASIC（CZ-8FB02）ではダンプ内容が異なるはずですが，心配する必要はありません。

```
MON
*D 2000 2020
:2000=0E 32 A7 0E C9 D1 3A 27 /.2ｧ.ﾉﾑ:'
:2008=36 B7 CA 45 20 2A 43 36 /6ｷﾊE *C6
:2010=22 34 36 2A 45 36 22 2E /"46*E6".
:2018=36 2A 41 36 C3 E5 15 3E /6*A6ﾃ.>
:2020=01 CD EC 0D CD 12 22 3E /.ﾍ ■.ﾍ.">
*■
```

```
*D 2000
:2000=0E 32 A7 0E C9 D1 3A 27 /.2ｧ.ﾉﾑ:'
:2008=36 B7 CA 45 20 2A 43 36 /6ｷﾊE *C6
:2010=22 34 36 2A 45 36 22 2E /"46*E6".
:2018=36 2A 41 36 C3 E5 15 3E /6*A6ﾃ♣.>
:2020=01 CD EC 0D CD 12 22 3E /.ﾍ▖.ﾍ.">
:2028=07 21 3E 01 21 3E 08 21 /.!>.!>.!
:2030=3E 0A 21 3E 09 21 3E 14 />.!>.!>.
:2038=21 3E 20 21 3E 21 21 3E /!> !>!!>
:2040=1A 21 3E 03 21 3E 11 21 /.!>.!>.!
:2048=3E 0F 21 3E 3F 21 3E 1E />.!>?!>.
:2050=21 3E 13 21 3E 17 21 3E /!>.!>.!>
:2058=0B 21 3E 06 21 3E 10 21 /.!>.!>.!
:2060=3E 0D 21 3E 23 21 3E 02 />.!>#!>.
:2068=21 3E 16 21 3E 22 21 3E /!>.!>"!>
:2070=05 18 03 CD 95 7F 21 00 /...ﾍ├π!.
:2078=00 22 CE 91 ED 7B 23 36 /."ﾎ|▞{#6
*■


:2070=05 18 03 CD 95 7F 21 00 /...ﾍ├π!.
:2078=00 22 CE 91 ED 7B 23 36 /."ﾎ|▞{#6
*D
:2080=4F 2A 34 36 23 7D B4 79 /O*46#}ｴy
:2088=28 22 2A 34 36 22 38 36 /("*46"86
:2090=22 30 36 2A 2E 36 22 3A /"06*.6":
:2098=36 2A 3E 36 22 3C 36 32 /6*>6"<62
:20A0=40 36 08 3A 26 36 3C FE /@6.:&6<生
:20A8=02 28 63 08 FE 4A 30 0E /.(c.生J0.
:20B0=11 C8 7B FE 26 38 0C 11 /.ﾈ{生&8..
:20B8=F7 7D D6 31 30 05 11 18 /日}ﾖ10...
:20C0=21 18 18 B7 28 F8 3D 28 /!..ｷ(時=(
:20C8=0D F5 1A 13 FE 80 28 03 /.火..生_(.
:20D0=B7 20 F7 F1 18 F0 1A FE /ｷ 日土.▒.生
:20D8=80 28 E3 3E 07 CD 13 00 /_(♥>.ﾍ..
:20E0=CD FA 1F CD A3 04 3A 26 /ﾍ秒.ﾍ｣.:&
:20E8=00 F5 3E 07 32 26 00 CD /.火>.2&.ﾍ
:20F0=0B 00 2A 34 36 23 7D B4 /..*46#}ｴ
:20F8=28 0D 2B 11 2A 21 CD 0B /(.+.*!ﾍ.
*■
```

図1.15 Dコマンド実行例

ダンプ画面の読み方は次の通りです。

アドレス	＋0	＋1	＋2	＋3	＋4	＋5	＋6	＋7	キャラクタ
:1000=	2B	73	C9	ED	78	B7	C8	C5	/+sﾉ▞xｷﾈﾅ

① 1行分（8バイト）の先頭アドレスが左端に16進4桁で表示される．

② ＝より右に，そのアドレス＋0，＋1，＋2，……，＋7番地の内容（各1バイト）が16進2桁で表示される．

③ ／(スラッシュ)より右には，各1バイトに対応するキャラクタ（8文字分）が表示される．

図1.16

／（スラッシュ）より右側に妙な文字列が表示されますが，これはメモリー内容各々を**キャラクタ・コード**とする文字（キャラクタ）です。X1シリーズにおいては，英数字・カナ・グラフィック文字に20H～FFHの1バイトコード（キャラクタ・コードとよぶ）を次のように割り当てています。

図1.17　キャラクタ・コード表

表では，キャラクタ・コードの上位4ビット（0～F）が縦方向，下位4ビット（0～F）が横方向に並べられています。たとえば，キャラクタ・コード41Hは，4の列を縦に，1の行を横に見て文字Aのコードであると読むわけです。

コード00H～1FHには，表示される文字ではなく，画面消去・ベル・改行などコントロール機能が対応しているので，これらのコードは**コントロール・コード**とよばれます。モニターのダンプ画面では，コントロール・コードの部分はすべて **.**（ピリオド）で表示しています。

では，モニターのダンプ画面におけるキャラクタの表示は何の役に立つのかと申しますと，メモリー内の「意味ある文字列データ」を探すのに使うのです。たとえば，次のように0F42H番地からをダンプしてみましょう（turboの方は，F531H番地から）（図1.18）。

キャラクタ・ダンプの部分を見ると，メモリー内のこの領域（エリア）には，ファンクション・キーのテーブル（表）が格納されていることがわかりますね。

以上で，**D**コマンドの使い方はすべてです。メモリーのいろいろな部分を**D**

コマンドでのぞいてみましょう！

```
*D 0F42
:0F42=06 46 49 4C 45 53 0D 00 /.FILES..
:0F4A=00 00 00 00 00 00 00 00 /........
:0F52=07 3F 54 49 4D 45 24 0D /.?TIME$.
:0F5A=00 00 00 00 00 00 00 00 /........
:0F62=03 4B 45 59 00 00 00 00 /.KEY....
:0F6A=00 00 00 00 00 00 00 00 /........
:0F72=06 4C 49 53 54 1A 0D 00 /.LIST...
:0F7A=00 00 00 00 00 00 00 00 /........
:0F82=06 52 55 4E 20 20 0D 00 /.RUN  ..
:0F8A=00 00 00 00 00 00 00 00 /........
:0F92=06 4C 4F 41 44 20 0D 00 /.LOAD ..
:0F9A=00 00 00 00 00 00 00 00 /........
:0FA2=06 57 49 44 54 48 20 00 /.WIDTH .
:0FAA=00 00 00 00 00 00 00 00 /........
:0FB2=05 43 48 52 24 28 00 00 /.CHR$(..
:0FBA=00 00 00 00 00 00 00 00 /........
*■
```

図1.18　ファンクション・キーのテーブル

1.7　メモリーへの書き込み（M コマンド）

次に覚えるコマンドは，メモリーにマシン語やデータを書き込むための **M** コマンドです。このコマンドは

　　＊ M　先頭アドレス

のようにして用います。たとえば，次のように入力してリターンキーを押すと，E000H 番地の現在の内容が表示され，カーソルが点滅します。

```
MON
*M E000
:E000=00
```

図1.19

では，この番地に 01H を書き込んでみましょう。01とキー入力し，リターンキーを押します。すると，次の番地である E001H 番地の内容が表示されます。

```
MON
*M E000
:E000=01
:E001=00
```

図1.20

以下，同様の操作を繰り返します。ここでは，次のようにキー入力していってください。

```
MON
*M E000
:E000=01
:E001=F4
:E002=31
:E003=3E
:E004=41
:E005=ED
:E006=79
:E007=01
:E008=F4
:E009=21
:E00A=3E
:E00B=02
:E00C=ED
:E00D=79
:E00E=C9
:E00F=00
```

図1.21

E00EH番地にC9Hを書き込んでリターンキーを押すと，E00FH番地に進みます。ここで，**M**コマンドから抜け出しましょう。そのためには SHIFT キーを押しながら BREAK キーを押してください。*が表示され，モニターのコマンドレベルに戻れます。

```
MON
*M E000
:E000=01
:E001=F4
:E002=31
:E003=3E
:E004=41
:E005=ED
:E006=79
:E007=01
:E008=F4
:E009=21
:E00A=3E
:E00B=02
:E00C=ED
:E00D=79
:E00E=C9
:E00F=00
*■
```

図1.22

では，キチンと書き込まれているか確認をします。D コマンドの使い方，よろしいですか。

```
*D E000 E00E
:E000=01 F4 31 3E 41 ED 79 01 /.ホ1>A▪▀y.
:E008=F4 21 3E 02 ED 79 C9 00 /ホ!>.▪▀yJ.
*■
```

図1.23　書き込んだ結果の確認

いかがですか？　予想通り，ダンプされましたか？

以上が M コマンドの使い方の基本なのですが，たくさんのデータを書き込みたい場合には，次のような便法が使えます。例として，先ほどと同じデータ列を E100H 番地から書き込んでみます。

今度は，1回ずつリターンキーを押すかわりにスペースを1文字分あけて，次々と入力していきます。C9H まで入力したら，初めてリターンキーを押します。

```
*M E100
:E100=01 F4 31 3E 41 ED 79 01 F4 21 3E 0
2 ED 79 C9
:E10F=00
*■
```

図1.24　M コマンドの便利な使い方(I)

ただし，画面の端にかかると見にくいですから，適当な位置でリターンキーを押しながら入力するとよいでしょう。

```
*M E100
:E100=01 F4 31 3E 41 ED 79 01 F4 21 3E
:E10B=02 ED 79 C9
:E10F=00
*■
```

図1.25　M コマンドの便利な使い方(II)

もう1つ，M コマンドの便利な使い方を紹介します。前節で見たファンクションキーの文字列のように，意味あるメッセージ列などをメモリー内に格納したいことがよくあります。そのためにキャラクタ・コード表を引いて文字のコードを調べて入力するのでは大変ですね。X1 のモニターでは，；(セミコロン)に続いて文字1字を書くと自動的にその文字のコードと見なしてくれます。たとえば；A は文字 A のコードである 41H を意味します。次の例をご覧ください。

```
*M E200
:E200=;T ;E ;S ;T
:E204=00
*D E200 E203
:E200=54 45 53 54 00 00 00 00 /TEST....
*■
```

図1.26　文字列データの格納

TEST という文字列データを E200H 番地から格納してみました。

以上で M コマンドの解説を終えますが，今回は D コマンドのときのように，

「いろいろ実験してください。」とは申しあげられないのです。というのは，メモリー内容の変更を勝手に行うとシステムを壊す危険があるからです。

X1 シリーズでは，**「クリーン設計」**といって，メモリーはすべて読み書き自由な **RAM（＝Random Access Memory）**にしてあり，BASIC システムも RAM 上に置かれています。この方法は，BASIC 以外の言語やシステムを使うときに大変便利なのですが，ユーザーがシステムを書き替えて壊してしまう危険も持っています。システムが壊れたら，テープやディスクから再びシステムをロードし直さなくてはなりません。

システムがメモリー内のどこに置かれているかは，後に詳しく学びますが，現在の段階では，D000H 番地〜E800H 番地のあたりで，**M** コマンドの実験をするのが安全であるとだけ申しあげておきます。

1.8 マシン語プログラムの実行（G コマンド）

前節までで私たちは，メモリー内にプログラムやデータを書き込み（**M** コマンド），それをダンプして確認する（**D** コマンド）ことができるようになりました。いよいよ，本節でプログラムの実行をしてみましょう。

前節において，私たちが E000H 番地から E00EH 番地まで15バイト分書き込んだデータ列は，実は「画面中央に文字 **A** を赤く表示させる」**マシン語プログラム**なのです。もう1度，ダンプしておきます。すでに消してしまった人や，次図のようになっていない人は，再度入力してください。

```
*D E000 E00E
:E000=01 F4 31 3E 41 ED 79 01 /・ホ1>A▪▀y・
:E008=F4 21 3E 02 ED 79 C9 00 /ホ!>・▪▀yJ・
*■
```

図1.27

確認できましたら，いよいよ実行にかかります。念のため SHIFT CLR/HOME を押して画面を消去し，SHIFT BREAK でモニターのコマンドレベルに戻っておくとよいでしょう。では次のように入力してリターンキーを押してください。

＊G E000

40桁画面のほぼ中央に文字Aが赤く表示されれば成功です。カーソルが点滅し，モニターのコマンドレベルに戻っていますか？

図1.28

私たちは初めて「マシン語プログラム」の実行を経験いたしました。「なぜ，これで文字Aが表示されるのか？」疑問百出かと思いますが，もう少しお待ちください。今は，ともかくモニター・コマンドの習得に励みましょう！

マシン語プログラムの実行に用いるモニター・コマンドはGコマンドです。

＊G 実行先頭アドレス

のようにして使います。先の例では，マシン語プログラムの実行がE000H番地から行われたわけです。プログラムのコピーをE100H番地にも入れてある方は，＊G E100としても同じ結果を得ますから実験してください。

1.9 メモリー内容の転送（Tコマンド）

前節の終わりでコピーのことに触れましたが，1度入力したプログラムやデータを別のアドレスにコピーするには，再度入力をしなくても，Tコマンドを使えば簡単にできます。Tはtransfer（トランスファ＝転送する）の頭文字をとったものです。

たとえば，E000H～E00EH番地の内容をE300H～E30EH番地にコピーするには，次のようにします。

```
*T E000 E00E E300
*D E300 E30E
:E300=01 F4 31 3E 41 ED 79 01 /.ホ1>A..y.
:E308=F4 21 3E 02 ED 79 C9 00 /ホ!>...y).
*■
```

図1.29　メモリー内容の転送

Tコマンドは，

　＊T　もとの先頭アドレス　もとの最終アドレス　転送先の先頭アドレス

という書式で用います。例では，E000H番地からE00EH番地までの内容をE300H番地以降に転送しています。

Tコマンドでは，もとの内容は消えずに残ることに注意しておきましょう。その意味では「転送」というより「コピー」の方が感じが出ますが，転送という用語もよく使われますので覚えておきましょう。

1.10　データの検索（Fコマンド）

HuBASICにはSEARCH（サーチ）という命令があって，BASICプログラム中の特定文字列を検索するのに便利ですが，これに相当するモニターコマンドがFコマンドです。

たとえば，0000H番地からFE00H番地の間のどこに，? TIMEという文字列が格納されているかを調べるには次のようにします。

```
*F 0 FE00 ;? ;T ;I ;M ;E
:0F53=3F 54 49 4D 45 /?TIME
*■
```

図1.30　文字列の検索

この例はX1のHuBASICでのものです（turbo BASICの方は多分F542H番地からとなるはずです）。

今の場合，0F53H 番地からをダンプすると，ファンクションキーのテーブル領域であることがわかります。

```
*D 0F53
:0F53=3F 54 49 4D 45 24 0D 00 /?TIME$..
:0F5B=00 00 00 00 00 00 00 03 /........
:0F63=4B 45 59 00 00 00 00 00 /KEY.....
:0F6B=00 00 00 00 00 00 00 06 /........
:0F73=4C 49 53 54 1A 0D 00 00 /LIST....
:0F7B=00 00 00 00 00 00 00 06 /........
:0F83=52 55 4E 20 20 0D 00 00 /RUN   ...
:0F8B=00 00 00 00 00 00 00 06 /........
:0F93=4C 4F 41 44 20 0D 00 00 /LOAD ...
:0F9B=00 00 00 00 00 00 00 06 /........
:0FA3=57 49 44 54 48 20 00 00 /WIDTH ..
:0FAB=00 00 00 00 00 00 00 05 /........
:0FB3=43 48 52 24 28 00 00 00 /CHR$(...
:0FBB=00 00 00 00 00 00 00 06 /........
:0FC3=50 41 4C 45 54 20 00 00 /PALET ..
:0FCB=00 00 00 00 00 00 00 05 /........
*■
```

図1.31

このように **F** コマンドは，BASIC システムを含めて未知のマシン語プログラムの解読をするときなどに威力を発揮します。**F** コマンドの書式は次の通り。

＊**F**　先頭アドレス　最終アドレス　データ　データ …

検索したい各データは16進２桁か，；に続くキャラクタ１字で与え，データ列の場合は，データ間に空白を１字ずつあけます。

1.11 メモリー内容の保存（S, L, V コマンド）

せっかく作成したマシン語プログラムやデータも，電源を切ると消えてしまいますから，カセットテープやディスクにセーブしておきましょう。モニターの **S**，**L**，**V** コマンドはカセットテープを対象としたコマンドです。（X1D や turbo ユーザーの方で専用カセットレコーダをお持ちでない方は，フロッピーディスクを用います。その方法は本節の終わりに述べます。）

1.7節でE000H番地からE00EH番地までに格納した「マシン語プログラム」(文字Aを赤く表示)をカセットテープにセーブしてみましょう(もう消してしまっている方は，1.7節の要領で再び入力しておいてください)。専用カセットレコーダに空テープをセットしてから，次のようにSコマンドを用います。

```
MON
*S E000 E00E E000:SAMPLE
```

図1.32 Sコマンドの使い方

上の指示は「E000H番地からE00EH番地までの内容を，実行アドレスE000H番地にして，テープにSAMPLEというファイル名でセーブする」と読みます。Sコマンドの書式は次の通り。

＊S　先頭アドレス　最終アドレス　実行アドレス：ファイル名

Sコマンドを実行すると，カセットレコーダが動き始めセーブが行われます。処理が終了するとテープが止まり，＊が表示されモニターのコマンドレベルに戻ります。

```
MON
*S E000 E00E E000:SAMPLE
Writing"SAMPLE          .    "
*█
```

図1.33

次に，実際にセーブされているか確認してみます。これにはVコマンドを用います。X1シリーズのように専用カセットレコーダの信頼性が高い場合はほとんど必要ない操作ですが，モニターコマンドの練習としてやってみましょう。

今，セーブを終えたテープを巻き戻し，頭出ししてから，次のように指示します。

```
*V :SAMPLE
```

図1.34 Vコマンドの使い方

(この場合，ファイル名を指定しましたが，正しく頭出しされているときはファイル名を省略して単に＊Vとして構いません。）リターンキーを押すとテープが動いて，テープ内のデータとメモリー上のデータが照合されます。正しい場合は次のようなメッセージが出ます。

```
*V :SAMPLE
Found  "SAMPLE          .   "
*■
```

図1.35

照合が正しく行われないときは **ERR?** が表示されテープは止まります。

正しくセーブされていたら，今度はロードをしてみましょう。実験のためにメモリー内をクリアします。X1のディスクBASICやturbo BASICで専用カセットレコーダを使用のユーザーは，ディスクよりBASICを起動し直してください。テープBASICをお使いの場合は，また読み込むのは大変ですから，「秘伝」をお教えします。モニターのGコマンドにより **＊G 14A0** を実行してください。

```
MON
*G 14A0
ト
Ok
■
```

図1.36 HuBASICのコールド・スタート

このようにすると，メモリー内がクリアされてからBASICのコマンドレベルに戻ります。

キチンとクリアされているか，念のため確認しておきましょう。

```
MON
*D E000 E00E
:E000=00 00 00 00 00 00 00 00 /........
:E008=00 00 00 00 00 00 00 00 /........
*■
```

図1.37 メモリー・クリアの確認

では，先にセーブしておいたプログラムをテープよりロードしてみます。テープの頭出しをした後，次のように操作します。

```
*L E000:SAMPLE
```

図1.38 Lコマンドの使い方

Lコマンドの書式は，

*L　ロード先頭アドレス：ファイル名

ですが，単に*Lとするとセーブしたときの情報でロードされます。今の場合は，キチンと頭出しできていれば，*Lだけで構いません。

さて，Lコマンドを実行すると次のメッセージが出ます。

```
*L E000:SAMPLE
Found  "SAMPLE
*■
```

図1.39

ロードが終わったら，E000H～E00EH番地をダンプして確認しましょう。

```
*D E000 E00E
:E000=01 F4 31 3E 41 ED 79 01 /.ホ1>A▪y.
:E008=F4 21 3E 02 ED 79 C9 00 /ホ!>.▪yJ.
*■
```

図1.40 ロードの確認

以上で，カセットテープとデータを出し入れするための3つのモニターコマンド（S, V, L）の解説は終わりです。

大切な宿題がありました。X1Dとturboのユーザーの方で，専用カセットレコーダをお持ちでない場合には，本節での実験ができません。ディスクとのセーブ，ロードについては，ほとんどの方がおわかりとは思いますが，念のため

にまとめておきます。

まず，E000H～E00EH 番地の内容を実行アドレス E000H にしてファイル名 **SAMPLE** でディスクにセーブするには，ディスク BASIC より次の命令を実行します。

```
*R
Ok
SAVEM "SAMPLE",&HE000,&HE00E,&HE000
Ok
■
```

図1.41 ディスクへのセーブ

HuBASIC の **SAVEM** 命令は，マシン語プログラムをセーブするためのもので，次の書式で指定します。

SAVEM 〝ファイル名〟，先頭アドレス，最終アドレス，実行アドレス

次に，ディスク内に格納されているマシン語ファイル（Bin ファイル）をロードするには **LOADM** 命令を用います。今の場合は，次のようにすれば **SAMPLE** というマシン語ファイルがロードされます。

```
LOADM "SAMPLE"
Ok
```

図1.42 ディスクからのロード

1.12 その他のモニター・コマンド

残ったモニター・コマンドは **P** と **R** です。このうち **R** コマンドについては，すでにご存知のはずですね。これは BASIC のコマンドレベルへ戻るためのコマンドです。

本節では，**P** コマンドの説明をします。このコマンドは，モニターの表示出力を画面にするか，プリンタにするかの切り替えを行うもので，プリンタをお持ちのユーザーのためのコマンドです。

プリンタを接続して電源ONの状態で次のように指示すると，出力がプリンタに切り替わり，画面にではなくプリンタの方にダンプリストが出力されます。

```
MON
*P
*D E000 E00E
*■
```

図1.43 Pコマンドの使い方

```
:E000=01 F4 31 3E 41 ED 79 01 /.ホ1>A■y.
:E008=F4 21 3E 02 ED 79 C9 00 /ホ!>.■y/.
```

図1.44 ダンプリストのプリンタ出力

Pコマンドを実行するたびに画面とプリンタが切り替わりますから，もう一度＊Pを行うと元に戻ります。

```
MON
*P
*D E000 E00E
*P
*D E000 E00E
:E000=01 F4 31 3E 41 ED 79 01 /.ホ1>A■y.
:E008=F4 21 3E 02 ED 79 C9 00 /ホ!>.■y/.
*■
```

図1.45

さあ，以上でモニターのコマンドはすべて使えるようになるはずです。わからない点が出てきたら，これからも適宜1.5節から本節までの内容を参照して，確実にマスターしましょう。

1.13 ニーモニック

私たちが初めて入力し実行したマシン語プログラムを思い出しましょう。それは16進表記で次のような姿をしていました。

```
:E000=01
:E001=F4
:E002=31
:E003=3E
:E004=41
:E005=ED
:E006=79
:E007=01
:E008=F4
:E009=21
:E00A=3E
:E00B=02
:E00C=ED
:E00D=79
:E00E=C9
```

図1.46

慣れた人にとっては，このような16進数の列を見ただけでプログラム内容が思い浮ぶようですが，恐らく多くの方にとっては「奇々怪々」というのが正直なところだろうと思います。また，CPU である Z80A に与える命令を16進数の形で憶えていられる人も数えるほどしかいないでしょう。そこで，CPU を開発したメーカーによって各マシン語命令には，**ニーモニック**とよばれる覚えやすい名称がつけられているのが普通です（ニーモニックは英語で mnemonic と書かれ,「記憶を助ける」という意味の言葉です）。Z80A は米ザイログ社によって開発されましたから,**ザイログ社仕様のニーモニック**が各命令についています。

本節冒頭で掲げたマシン語プログラム中の各命令は，次のようなニーモニックと対応します。

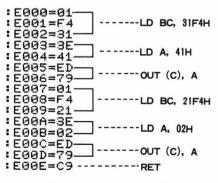



図1.47　ニーモニックとの対応

Z80Aの場合，1つのまとまった意味をもつマシン語命令は1バイト～4バイトで表わされます。そのまとまりごとにニーモニックが対応しています。上の例では，C9Hは1バイト命令で**RET**とニーモニック表記され，**3EH, 41H**は2バイト命令で**LD A, 41H**と対応，**01H, F4H, 31H**は3バイト命令で**LD BC, 31F4H**と対応するといった具合です。

このようにニーモニックは，各マシン語命令の内容を連想させる英単語あるいはその省略形からできています。**LD**は**load**の略で「転送命令」を意味します。**RET**はreturnの略で，BASICと同様に(マシン語の)サブルーチンからの復帰命令を意味しています。各ニーモニックのキチンとした意味は本書全体で勉強することにして，ここではニーモニックからの「連想」を頼りに，対応する処理をBASICで記述してみることにします。次のリストがそれです。

```
10 BC=&H31F4
20 A=&H41
30 OUT BC,A
40 BC=&H21F4
50 A=2
60 OUT BC,A
70 RETURN
```

図1.48 対応するBASICプログラム(I)

いかがですか？ マシン語プログラムがどんな処理をしているか，何となくおわかりいただけましたか？ では，このBASICプログラムを実行してみましょう。サブルーチンですから，**RUN**ではなく**GOSUB 10**により走らせます。

```
LIST
10 BC=&H31F4
20 A=&H41
30 OUT BC,A
40 BC=&H21F4
50 A=2
60 OUT BC,A
70 RETURN
Ok
GOSUB 10
Ok

                              A
```

図1.49 BASICプログラム(I)の実行

画面中央に文字**A**が赤く表示されれば，実験成功です。

「BASICならば，こんな面倒なプログラムにしなくても……」という声が聞こえそうです。その通りです。全く同じ処理は，BASICで次のように書けます。

```
10 LOCATE 20,12
20 COLOR 2
30 PRINT "A"
40 RETURN
```

図1.50　対応するBASICプログラム(II)

念のため実行してみます（ただし，**COLOR2**のため**OK**表示が赤くなる)。

```
LIST
10 LOCATE 20,12
20 COLOR 2
30 PRINT "A"
40 RETURN
Ok
GOSUB 10
```

図1.51　BASICプログラム(II)の実行

これら2つのプログラムの違いが，マシン語によるプログラミングの特徴をかなり伝えてくれます。すなわち，文字**A**を表示するのに，BASICでは**PRINT**命令でよかったわけですが，マシン語プログラムにおいては，「**A**を表示する処理はX1でどのようにするか」ということ自体をCPUに指示しなければなりません。そのためには，本章の最初に述べたようなコンピュータの動作のしくみやX1シリーズの機械的しくみ(**ハードウェア**)についての知識も必要になるのです。「マシン語が難しい」といわれる原因の1つは，この辺の事情にあるような気がしますが，逆にマシン語でX1シリーズを操作できるようになると，「愛機の隅々までわかる」ようになります。皆さんの愛機をトコトン理解するためにも，マシン語の習得へ向けて頑張りましょう。

1.14 アセンブラ

ニーモニックで書かれたマシン語プログラムを**ソースプログラム**とよびます。これは人間にとっては処理内容がわかって便利なのですが，**LD A, 41H**などと指示してもCPUにはチンプンカンプンです。CPUにわかるのはあくまでメモリー内に格納された**ビットパターン**です。私たちには，モニターなどを通じて16進数の列として写るわけですが，今後，「コード」としての面を強調して，**マシンコード**という用語を本書では使います。

［例］

ニーモニック		マシンコード
LD A, 41H	↔	**3EH 41H**

こうして，私たちがマシン語プログラムを作成する段階ではニーモニック表記を用い，ソースプログラムをマシンコードに翻訳してから，CPUに実行させるという手順を踏むことになります。この翻訳操作のことを**アセンブル(assemble)**とよびます。ソースプログラムをアセンブルしてできたマシンコードからなるプログラムを**オブジェクトプログラム**あるいは**オブジェクト(object)**といいます（objectは「対象，目的」を意味する英語です)。

ニーモニックとマシンコードは1対1に対応していますから，巻末付録の命令表を用いれば人間の手でも，アセンブル作業をすることは可能です。これを**ハンド・アセンブル**といいます。しかし，ソースプログラムの長さが長くなると，気の遠くなるような作業になるので，アセンブル作業自体もプログラム化してコンピュータにさせることを考えます。そのための翻訳プログラムを**アセンブラ(assembler)**とよびます。X1シリーズにおいても，すでにいくつかのアセンブラが市販されるようになりました。

アセンブラが翻訳処理の対象とするのは，ニーモニック表記されたマシン語命令の列ですが，これら以外にオブジェクトを何番地から格納するのかなどの指示もしなくてはなりません。**アセンブラ指示命令（疑似命令）**とよばれる一群の命令がそれです。ニーモニック表記のマシン語命令とアセンブラ指示命令

を合わせた総体を**アセンブリ言語（assembly language）**といいます。

アセンブラ指示命令の部分は個々のアセンブラで少しずつ異なるようですが，比較的多くのアセンブラで用いられているアセンブリ言語により，例の「文字**A**を表示する」プログラムを記述したのが次のリストです。

```
ORG  0E000H
LD   BC,31F4H
LD   A,41H
OUT  (C),A
LD   BC,21F4H
LD   A,02H
OUT  (C),A
RET
END
```

図1.52　ソースプログラムの例

ORGと**END**というのがアセンブラ指示命令です。**ORG 0E000H**は，オブジェクトをE000H番地から作成するようアセンブラに指示します。また，**END**はソースプログラムの終わりを示し，アセンブラにアセンブル作業の終了を指示します。

第2章　Z80 CPUとハンド・アセンブル

本章では，X1（およびturbo）をマシン語で動かしながら，学習を進めていく上で必要なメインCPU **Z80A**に関する簡単なハードウェアの知識（バスやレジスタ構成，I/Oポートなど）と，ハンド・アセンブルの具体的方法について学びます。(X1では，Z80A以外にもキーボードやテレビなどの制御用にCPUが使われているので，Z80Aを**メインCPU**，他を**サブCPU**とよんで区別します。)

Z80あるいはZ80Aについては，本来説明しなければならないことは，もっとたくさんありますが，本書では思い切って，「マシン語初心者の方がX1を操作する上で必要なZ80Aの知識」に限定いたしました。したがって，Z80に関してだけならば，本書の説明はかなり不十分なものですが，さいわい，Z80 CPUは8ビット・パソコンや制御機器に多く使われている関係で，良い解説書も豊富にあります。これらについて，より詳しく知りたい読者は以下に掲げる参考文献を参照してください。また，本文中でも何か所か下記文献を参照しています。その際，[　]で囲まれた数字が文献の番号に対応しています。

まず，㈱シャープより発表されている「公式マニュアル」類が何冊かあります。そのうち本書で用いる基礎文献として次を挙げておきます（以下[　]内の数字は本書で参照する文献番号)。

[1]　Z80ファミリーテクニカルマニュアル[I](シャープ発行，エレクトロニクスダイジェスト書店部発売)

上記のマニュアルが入手しにくい場合には，次の本がコンパクトにまとめられていて便利かと思います。

[2]　図解マイクロコンピュータZ-80の使い方（横田英一著。オーム社刊)

また，Z80 CPUの各マシン語命令の使い方に関しては，次の本が懇切丁寧で初心者向きでしょう。

［3］ Z80マイコンプログラムテクニック（庄司 渉，本田 稔著。電波新聞社刊）

2.1 Z80 CPUの端子

Z80 CPUは，1975年にアメリカのザイログ社から発表された8ビットの**マイクロプロセッサ**（コンピュータのCPU機能を有するLSI）で，先行して市場に出回っていたインテル社の8080 CPUの機能をすべて含み，さらに豊富な拡張機能を持っていたため，広く普及するようになりました。

マイクロプロセッサは，水晶発振器から出される規則的なクロック信号により動作しますが，Z80の場合，クロック信号の最大周波数が異なるいくつかの同型CPUがあり，Z80A，Z80B，Z80Hなど後続する英記号で区別されます。たとえば，単なるZ80はクロック周波数の最大値が2.5MHzであるのに対し，Z80Aは4MHzのクロック周波数で動作できます（MHzはメガヘルツと読み，1秒間に100万回の振動を表わす単位です）。クロック周波数が大きいほど高速動作が可能なわけで，Z80B，Z80Hはさらに「高速化」されています。

X1シリーズ（turboを含めて）では，Z80Aを4MHzの最大周波数で動作させています。ただし，これらZ80 CPUの高速版は，クロック周波数に関する機械的部分以外では機能が同じなので，本書では簡潔のためZ80AとZ80を区別せず，単にZ80として扱うことにします。

さて，Z80 CPUは40個の端子を持ったLSIです。その端子配置図を掲げます（図2.1）。

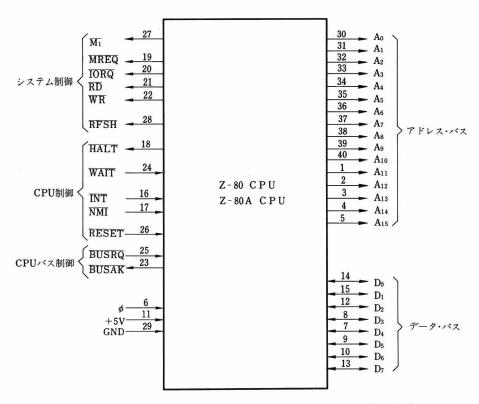
これらの全端子の意味を覚えるのは大変ですから，知りたい方は文献［1］や［2］で自習していただくことにして，次節ではアドレス・バスとデータ・バスについて学びましょう。

（注） 端子につけられた数字は端子番号で，実際には，この番号順に端子が並んでいる．

図2.1 Z80 CPU の端子配置図

2.2 アドレス・バスとデータ・バス

コンピュータ本体の基盤上にはマイクロプロセッサを中心として，メモリーICやさまざまな周辺LSIなどが整然と配置されています。これらの素子たちは数多くの信号線で互いに結ばれていますが，これを「道路」にたとえるなら，その中でも中心的な「高速道路」に相当するものが**システム・バス（system bus）**とよばれる信号線の束です。CPU，メモリー，入出力用LSIなどはシステム・バスに「ぶらさがって」，システム・バスはこれら共用の信号線として使

われます。

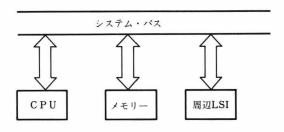



図2.2 システム・バス

Z80 CPUを用いたコンピュータでは，システム•バスはその働きにより，さらにアドレス・バス，データ・バス，制御バスの3種類に分けられます。

アドレス・バスは16本の信号線からなり，メモリーや入出力装置に番地（アドレス）を指定します。各信号線は A_0 から A_{15} までの名称で区別され，アドレス指定の16ビット値とは次のように対応します。

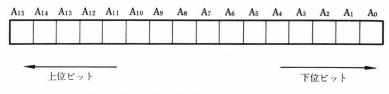



図2.3 アドレス・バスとアドレス値

たとえば，CPUがメモリーのFF00H番地と情報のやりとりをするときには，CPUは A_{15}～A_8 の信号線を1（電圧の高い状態），A_7～A_0 の信号線を0（電圧の低い状態）にして，メモリーにその番地を知らせるのです。

データ•バスは D_0～D_7 とよばれる8本の信号線からなり，CPU，メモリー，入出力装置間でやりとりされる8ビットデータがその上を往来します。アドレス•バスはCPUから周辺へ向かう**単方向のバス**なのに対して，データ•バスは**双方向のバス**であることが特徴です。

データ・バスの各信号線と，8ビットデータとの対応関係は次図のようになっています。

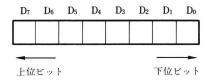



図2.4 データ・バスとビット位置

2進数で2^nの位にあたるビット位置を**第nビット**とか**ビットn**とよびますが，その用語を使うと，データ・バスのD_nは1バイトデータのビットnに対応しているわけです。

CPUがメモリーや入出力装置と情報のやりとりをする（**アクセスする**といいます）には，アドレス・バスに番地指定の信号を出してから，データ・バスを通じデータが流れることになります。その際，「アクセスするのはメモリーか入出力装置（I/O）か」の区別や，「データはCPUから出力されるのか，CPUに入力されるのか」というデータ・バスの方向性は，**制御バス**により指定されます。制御バスの働きは，CPUが動作するタイミングとも関係して複雑ですから，本書ではこれ以上述べません。[1]や[2]の文献を参照してください。

2.3 I/Oポート

入出力装置というと頭に浮かぶのは，ディスプレイ装置やプリンタなどですが（この2つは出力装置），注意すべきことは，これらの装置が直接にシステム・バスを通じてCPUとつながれているのではない！という点です。CPUは微小電圧の電気信号をやりとりしていて，これらの入出力装置とは電圧レベルにおいても，またタイミングにおいても大きく異なっています。

そこで両者の間に入って，一方から他方に電気信号を変換する回路が必要となります。これを**インターフェース**とよんでいます。

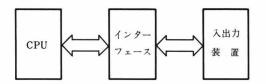



図2.5 インターフェース

複雑なことはすべてインターフェース側に任せて，これを「ブラックボックス」と見なしてしまえば，CPU から見たインターフェースは入出力信号の単なる出入口ということになります。これを**入出力ポート**あるいは**I/O ポート**とよびます（port とは，出入口や港を意味する英語です）。

前節で，CPU が入出力装置をアクセスすることに触れましたが，正しくは I/O ポートをアクセスするのです。各ポートは，メモリーと同様に番地で区別され，アドレス・バスにより指定されます。I/O ポートのアドレスを**I/O アドレス**とよびます。

Z80 CPU のふつうの使い方では，I/O ポートを指定するのに，アドレス・バスの上位 8 ビット (A_{15}〜A_8) は無視することになっています。後の第 6 章でも触れますが，Z80 の入出力命令のニーモニック表記にも，こうした使い方が反映されています。しかし，文献[1]を見ると，Z80 の入出力命令では A_{15}〜A_8 にも信号が出力されており，実際，X1 シリーズ (turbo を含め) では Z80 をそのように使っています。このような理由から，X1 シリーズでは I/O アドレスがメモリーと全く同じ 0000H〜FFFFH の65536個も存在しています。

Z80 CPU の多くの解説書で書かれている「I/O ポートは256個」という記述と，この点で異なっていますから注意してください。

2.4 レジスタ構成

Z80 CPU は，合計207ビット分の記憶回路を内部に持っていて，私たちユーザーがプログラムにより利用できるようになっています。これらもメモリーの一種ですが，CPU 内部にあるのでとくに**レジスタ**（register）とよばれます。

メモリーのアクセスは，システム・バスを通じて行なわねばなりませんが，レジスタのアクセスはCPU内部で処理されるので高速に行なえます。

Z80のレジスタ構成は，17個の8ビット・レジスタ，4個の16ビット・レジスタ，1個の7ビット・レジスタに分かれています。それらを名称とともに次に掲げます。

主レジスタ・セット		補助レジスタ・セット		
アキュムレータ A	フラグ F	アキュムレータ A′	フラグ F′	
B	C	B′	C′	汎用レジスタ群
D	E	D′	E′	
H	L	H′	L′	

割り込みベクトル I	メモリ・リフレッシュ R	専用レジスタ群
インデックス・レジスタIX		
インデックス・レジスタIY		
スタック・ポインタSP		
プログラム・カウンタPC		

（注）Rレジスタは7ビット・レジスタ

図2.6 Z80 CPUのレジスタ構成

2.5 プログラム・カウンタ

本節ではまず，CPUの動作の基本となる**プログラム・カウンタPC**について説明します。

第1章1.1節で述べたように，現在のフォン・ノイマン式のコンピュータでは，CPUはあらかじめメモリー内に格納しておいたデータ列を「マシン語命令」と見なして，逐次処理するように設計されているのでしたね。このとき，CPUが今メモリーの何番地に注目しているかが決定的に大切です。Z80 CPUにおい

て，この役割を担っているのがプログラム・カウンタであるわけです。

プログラム・カウンタ（PC）は，16ビットのレジスタで，CPUが現在実行中の命令のメモリー番地16ビットを保持しています。

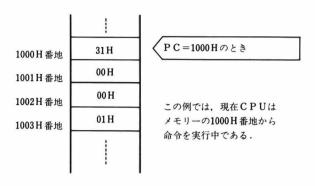



図2.7 プログラム・カウンタ（PC）

Z80 CPUは，PCの内容16ビットをアドレス・バスに送り出すことで，メモリー番地を指定し，その番地の内容を「命令」として取り込みます。ふつうは，この後自動的にPCの内容が1増加されCPUは次の番地に注目する，というように動作します。これはちょうどBASICプログラムが原則として行番号の順に実行される様子と同様です。

ただ，BASICでは**GOTO**や**GOSUB**でプログラムの流れを変えられるのと同様に，マシン語レベルでもCPUが後述する「ジャンプ命令」(第5章参照)を実行すると，新しいアドレス値が直接PCにセットされ，CPUは指定された新しいメモリー番地に制御を移します。

2.6 アキュムレータ

さて，プログラム・カウンタ（PC）はCPUの動作に直接関係するレジスタでしたが，私たちがマシン語プログラムを作ろうとするとき，最初に覚えなくてはならないレジスタはAレジスタです。

Aレジスタは，**アキュムレータ（accumulator）**ともよばれる8ビットレジス

タです。このレジスタは，メモリーやI/Oポートとのデータのやりとり，CPUの行う様々な演算処理で中心的な役割を果たします。

たとえば，メモリーのD010H番地にデータ40Hを書き込む処理をCPUにさせるにはどうすればよいのでしょう？

まずBASICでプログラムしてみます。メモリーにデータを書き込むためのBASIC命令は**POKE**です。次のプログラムを**RUN**してください。

```
LIST
10 POKE &HD010,&H40
20 END
Ok
RUN
Ok
■
```

図2.8 BASICプログラム

結果は，モニターを起動しD010H番地をダンプすれば確認されます。

```
MON
*D D010
:D010=40 00 00 00 00 00 00 00 /@.......
:D018=00 00 00 00 00 00 00 00 /........
:D020=00 00 00 00 00 00 00 00 /........
:D028=00 00 00 00 00 00 00 00 /........
:D030=00 00 00 00 00 00 00 00 /........
:D038=00 00 00 00 00 00 00 00 /........
:D040=00 00 00 00 00 00 00 00 /........
:D048=00 00 00 00 00 00 00 00 /........
:D050=00 00 00 00 00 00 00 00 /........
:D058=00 00 00 00 00 00 00 00 /........
:D060=00 00 00 00 00 00 00 00 /........
:D068=00 00 00 00 00 00 00 00 /........
:D070=00 00 00 00 00 00 00 00 /........
:D078=00 00 00 00 00 00 00 00 /........
:D080=00 00 00 00 00 00 00 00 /........
:D088=00 00 00 00 00 00 00 00 /........
*■
```

図2.9 結果の確認

では，これと同じことをマシン語で行ってみましょう。CPUの立場になって考えてみると，BASICの**POKE**文で実現された処理は2段階に分かれます。

(CPU が1度にできることは非常に単純なことなのです！)

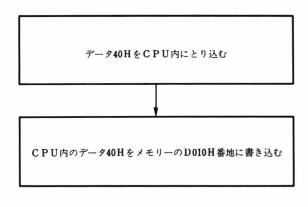



図2.10

ここでデータ 40H が取り込まれる先が，CPU 内の A レジスタであるわけです。上の 2 つの処理はアセンブリ言語で次のようにニーモニック表記されます。

```
LD  A, 40H
LD  (0D010H), A
```

LD A, 40H は「A レジスタにデータ40H をロードする」を意味し，LD (0D010H), A は「メモリーの D010H 番地に A レジスタの内容をロードする」という意味です。ここで LD の部分が「ロード (load)」の省略形で，「(データを) 転送する」を意味しています。ニーモニック表記で LD のつく命令は「**ロード命令**」とか「**転送命令**」とよばれます。

2.7　ハンド・アセンブルの方法

では，前節のプログラムを実行しましょう。といっても，ニーモニック表記では CPU が理解してくれませんから，これをアセンブルしてオブジェクトのマシンコードを生成しなければなりません。アセンブラをお持ちの方は，それを用いてくださって結構です。本書では，アセンブラをまだお持ちでない初め

ての方のために**ハンド・アセンブル**を敢えて実行します。アセンブラをお持ちの方もハンド・アセンブルに慣れておくと，マシンコードの理解のためによいと思います。

まず，上に登場した2つの命令の一般形を書くと次のようになります。

LD A, n	Aレジスタに1バイトデータnをロードする．
LD (mn), A	mを上位バイト，nを下位バイトとしてアドレス指定されるメモリー番地にAレジスタの内容をロードする．

※例では，LD A, nでn＝40Hとした．
※例では，LD (mn), Aでm＝D0H，n＝10Hとした．

図2.11

次に，付録1「Z80命令表」の8ビットロード命令の項を見ますと以下のことがわかります。

マシンコード（16進）	ニーモニック
3E n	LD A, n
32 n m	LD (mn), A

図2.12 ニーモニックとマシンコードの対応

オブジェクトでは，LD A, nは2バイト命令，LD (mn), Aは3バイト命令になっています。

こうして「ハンド・アセンブル」により，例のプログラムのマシンコードは次のようになるはずです。

マシンコード（16進）	ニーモニック
3E 40	LD A, 40H
32 10 D0	LD (0D010H), A

図2.13　例のプログラムのオブジェクト

では，メモリー上D000H番地から，これらのオブジェクトを格納してみましょう。モニターを起動し，**M**コマンドで次のように入力します（このように1つのニーモニックに対応する数バイトごとに改行していくと誤りが少ないと思います）。

```
MON
*M D000
:D000=3E 40
:D002=32 10 D0
:D005=00
*■
```

図2.14

さあ実行！といきたいのですが，何かを忘れていませんか？ CPUがD002H番地から始まる3バイト命令を解釈した後にプログラム・カウンタ(PC)はD005Hを示しているはずですね。そしてCPUはD005H番地からのデータを「命令」として解釈・実行していきます。ということは……そう！マシン語のプログラムの最後には何らかの「実行停止処置」をしておかないとダメなのですよ。

私たちは，今，マシン語プログラムをモニターより**G**コマンドで実行することを想定しています。この場合，「再びモニターへ戻ることで実行停止をする」という方法が使えます。そのための「オマジナイ」を1つ覚えておきましょう。それは，プログラムの最後にマシンコードC9Hを書いておくのです。この命令は「リターン命令」といって，ニーモニック**RET**で表記される1バイト命令です。以上をまとめて，D000H番地から配置し，モニターの**G**コマンドで実行可

能な形のオブジェクトは，次のようになります。

アドレス	マシンコード	ニーモニック
D000	3E 40	LD A, 40H
D002	32 10 D0	LD (0D010H), A
D005	C9	RET

図2.15 完成したオブジェクト

これを **M** コマンドで書き込みます。また，実験に使われる D010H 番地の内容を 00H に戻しておきましょう。

```
MON
*M D000
:D000=3E 40
:D002=32 10 D0
:D005=C9
:D006=00
*M D010
:D010=00
:D011=00
*■
```

図2.16

いよいよ実行にかかります。**＊ G D000**でしたね。モニターに戻れば成功です。結果を見るため D000H 番地からをダンプしてみます (図2.17)。

図のようなダンプリストであれば，実験成功です。D000H 番地から D005H 番地まではプログラムの部分です。実行結果は D010H 番地にあります。ここが，40H になっていれば予想通り！ですね。万一，うまくいかなかった方は，本節最初からの説明を読み直してもう１度アタックしてください。

こうして私たちは，

① ニーモニック表記でソースプログラムを書く。

② ハンド・アセンブルにより，マシンコードに翻訳し，オブジェクトを生成する。

```
*G D000
*D D000
:D000=3E 40 32 10 D0 C9 00 00 />@2.ミノ..
:D008=00 00 00 00 00 00 00 00 /........
:D010=40 00 00 00 00 00 00 00 /@.......
:D018=00 00 00 00 00 00 00 00 /........
:D020=00 00 00 00 00 00 00 00 /........
:D028=00 00 00 00 00 00 00 00 /........
:D030=00 00 00 00 00 00 00 00 /........
:D038=00 00 00 00 00 00 00 00 /........
:D040=00 00 00 00 00 00 00 00 /........
:D048=00 00 00 00 00 00 00 00 /........
:D050=00 00 00 00 00 00 00 00 /........
:D058=00 00 00 00 00 00 00 00 /........
:D060=00 00 00 00 00 00 00 00 /........
:D068=00 00 00 00 00 00 00 00 /........
:D070=00 00 00 00 00 00 00 00 /........
:D078=00 00 00 00 00 00 00 00 /........
*■
```

図2.17

③ メモリー上にオブジェクトを配置する。

④ オブジェクトを実行する。

という一連の手順を初めて体験したことになるわけです。これから私たちのマシン語学習は，この手順を何度も繰り返していきます。

2.8 ニーモニック表記のルール

本節では，ザイログ社仕様の Z80 ニーモニックの構造と，その表記上のルールについてまとめておきましょう。

前節で登場した命令を例に説明します。

```
LD A, n
```

このニーモニック表記は大きく 2 つの部分に分けられます。まず **LD** の部分は，この命令が「ロード命令」であることを示していて，**オペレーション・コード** (operation code) あるいは略して**オペコード**，**OP コード**などとよばれます。後半の **A, n** の部分は，命令の対象がどこなのかを示しており，**オペランド** (operand) とよびます。今の場合，オペランドは **,** (カンマ) で区切られた

2つの部分にさらに分けられ，前者**A**を**第1オペランド**，後者**n**を**第2オペランド**といいます。以上をまとめると次のようになります。

LD	A	,	*n*
OPコード	第1オペランド		第2オペランド
	オペランド		

これがZ80ニーモニックの基本構造です。ただ，命令によってはRETのようにOPコードのみでオペランドを持たないものや，オペランドが第1オペランドだけしかないものもあります。

次に，ニーモニック表記上（一般的なZ80アセンブラを使うとき）覚えておくべきルールについて説明します。

オペランドに数値が登場するときは，10進表記でも16進表記でも構いませんが，16進表記のときには，末尾にHをつけます。何もつけないと，その数値は10進数と見なします。

［例］　LD　A, 10
　　　　（Aレジスタに10進数10をロード）

オペランド内に16進数を書くとき，それがA，B，C，D，E，Fのいずれかで始まるならば，その前にさらに0（ゼロ）をつけ加えます。これは，用いられる英字がレジスタ名と共通なので，16進数字の方であることを明記するための配慮です。

［例］　LD　A, 0D1H
　　　　（Aレジスタに16進数D1Hをロード）

オペランド内の括弧（　　）は，その内部の数値，レジスタなどが示すメモリー番地の内容を意味します。たとえば，

LD　(0D010H), A

において，(0D010H) は16ビット値 D010H 自体ではなく，メモリーの D010H 番地の内容を示します。

最後に細かいことですが，ニーモニック表記において，OP コードとオペランドの間は空白1文字以上あけるのが原則です。

[例] LDA, 40H ……よくない例
　　 LD ␣ A, 40H ……正しい例
　　 空白

さらにもう少し欲を出して「見やすいソースリストを書く」点から述べますと，Z80 ニーモニックのOP コードは4文字以内になっているので，オペランドの先頭をそろえるために，オペランドを（ニーモニック先頭から数えて）6文字目から書き始める習慣をつけるとよいと思います。

[例] LD ␣␣␣ A, 40H
　　 空白

2.9 上下位逆転の原則

2.7節において初めてハンド•アセンブルを体験いたしましたが，そのとき1つの疑問を抱かれた皆さんも多いと思います。それはおそらく LD (mn) , A のアセンブルに関することだと推察します。もう1度思い出しましょう。

マシンコード	ニーモニック
32 n m	LD (mn), A

図2.18

前に見逃した方も，ここで「アレッ？」と思いませんか？　マシンコードでnとmが逆転していますね。これは誤植ではないのです。

Z80では（さらに広く80系のCPUでは），ニーモニック内に2バイト数値が指定されているとき，アセンブル時にはその上位バイトと下位バイトを逆転してオブジェクトを生成するという原則があります。これを「**上下位逆転の原則**」とよんでおきましょう。

今の場合は，第1オペランドにmnという2バイト数値が含まれていますから，マシンコードに変換するときに下位バイトnを先に，上位バイトmを後にしなくてはならないのです。

6809のような68系のCPUではこのようなことはなく，「上下位逆転の原則」は80系CPUの特徴になっています。この是非は別として，X1シリーズがZ80Aを搭載している以上，私たちは「上下位逆転の原則」と直面せざるを得ないのですから，ともかく慣れることにしましょう。

以上により，**LD　(0D010H),A** のマシンコードが，32 10 D0の順になることは了解いただけたと思います。

第3章　データのロード（転送）

本章より，Z80 CPUの個々の命令とその使い方を見ていきましょう。命令の中には対象とするレジスタが違うだけのものなども含まれていますから，本書では「命令の働き」に注目し，よく使われる命令，面白い命令を選んでアラカルト風に紹介していきます。各記述は「実験」精神を重んじて，実際にX1（あるいはturbo）で確認しながら進めていきます。ほとんどが短いプログラムですから，皆さんも是非，入力して実験してみてください。今まで遠いものに思われていたマシン語が，きっと身近な存在となることでしょう。

さて，本章ではその先頭を切って，マシン語命令の中でも最も基本的な**ロード命令**について扱います。これは，ある所のデータを他の所へ「コピー」する命令です。慣用として，「転送命令」という言い方もされます。私たちの日常語感では，「転送」というと実際にものが動いて，もとの場所にはなくなってしまうと考えられますが，本書では「転送」を「ロード」の訳語とし，「コピー」の意味あいで使います。したがって，もとの場所のデータは保持されます。

厳密には，「ロード」というのはCPUに遠い方（たとえばメモリー）から，CPUに近い方（たとえばレジスタ）へデータを移動させるのに使うようです。逆に，CPUに近い方から遠い方への移動には「ストア」（格納）という言い方もされます。Z80の先祖8080のニーモニックではこれらを区別していますが，Z80ではすべて**ロード（OPコードLD）**で統一しています。

Z80のロード命令には，8ビット・データを対象とするもの，16ビット・データを対象とするものがあります。このほかに，スタックとよばれる特別なメモリー領域とレジスタ間のデータ転送を行うプッシュ（**PUSH**），ポップ（**POP**）命令も含めて考えることにします。

本章で述べられる内容は，以後の各章で基本テクニックとなるものばかりですから，じっくり時間をかけて一歩一歩着実に理解するようにしましょう。

3.1　8ビット・ロード命令

LD A, n や LD (mn), A などは8ビットデータの転送(ロード)に関するもので，**8ビット・ロード命令**と総称されます。これらの命令のリストは付録1の命令表を参照してください。

8ビット・ロード命令のニーモニックでは，OPコードはすべてLDで統一されています。オペランドは必ず2つ存在し，第1オペランドは転送先を，第2オペランドは転送元（送り出し側）を指定します。転送先のことを**デスティネーション（destination＝目的地）**，転送元のことを**ソース（source＝源）**といいます。

LD　デスティネーション，ソース

図3.1　8ビット・ロード命令のニーモニック

次節からは，まず「命令表」を読むために必要な2つの知識を学んでおきましょう。レジスタ・ペアとインデックス・レジスタについてです。

たとえば，LD A, (HL)という命令がありますが，これはどう読めばよいのでしょう。括弧(　)の意味は，すでに知っていますが，その中味であるHLとは何でしょう。

3.2　レジスタ・ペア

Z80内部にある8ビットのレジスタ群のうち，**主レジスタ**とよばれるA, F, B, C, D, E, H, L（および対応する補助レジスタA′, F′, B′, C′, D′, E′, H′, L′)は特定のレジスタどうしを2つつないで16ビット・レジスタとして扱うことが可能です。これを**レジスタ・ペア**とよびます。レジスタ・ペアには次のものが許されています。

主レジスタ	補助レジスタ
AF (=AとF)	AF′ (=A′とF′)
BC (=BとC)	BC′ (=B′とC′)
DE (=DとE)	DE′ (=D′とE′)
HL (=HとL)	HL′ (=H′とL′)

図3.2 レジスタ・ペア

各レジスタ・ペアで16ビット値を扱うとき，上位1バイトを受け持つレジスタ，下位1バイトを受け持つレジスタは明確に決まっています。たとえば，HLレジスタ・ペアでは，上位バイトはHレジスタに，下位バイトはLレジスタにと決められていて，その逆はできませんから注意しましょう（もともとHとLは，その意味で名称がつけられていて，Hはhighの頭文字，Lはlowの頭文字です）。

以上のことを理解すると，**LD A,(HL)**の意味がわかります。すなわち「HLレジスタ・ペアの示すメモリー番地の内容を**A**レジスタにロードせよ。」という命令になります。

3.3 インデックス・レジスタ

ではもう1つ，**インデックス・レジスタ**について学んでおきます。IX，IYとよばれる2本の16ビット・レジスタがそれです。これらのレジスタは，16ビットのアドレス値を指示するために（主に）用いられます。使い方を説明するのに1つの例を挙げましょう。

LD A,(IX+d)

という8ビット・ロード命令があります。今，**IX**レジスタの内容がE000Hであったとしましょう。このとき，**(IX+d)**とは，「**IX**レジスタの内容であるE000H

にdを加えた16ビット値の示すメモリー番地の内容」を意味します。dは1バイト数値で，10進数表記すると－128～＋127の範囲の「符号つき整数」です(詳しくは第4章4.1節参照)。IXの示す基準番地からの隔たりを表わすので**ディスプレイスメント（displacement＝変位）**とよばれます。

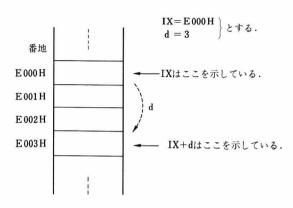



図3.3 インデックス・レジスタ

こうして，LD A,(IX＋d)は「IX＋d番地の内容をAレジスタにロードせよ。」という意味だとわかりますね。

このようにインデックス・レジスタは基準アドレス値とそこからの隔たりという形で，メモリー上のデータを扱うのに適しており，BASICでいえば「配列変数」に相当するものをマシン語で利用したいときなどに用いると便利です。

以上の知識を総合すれば，8ビット・ロード命令のすべてのニーモニック表記の意味が理解できるはずです。ただし，特殊な2つのレジスタであるIレジスタ（割り込みベクトル・レジスタ），Rレジスタ（メモリー・リフレッシュ・レジスタ）に関係したロード命令については文献［1］，［2］などに譲ります。

［注］ Iレジスタは，第10章10.10節で登場します。

3.4 16ビット・ロード命令

Z80 CPUでは，8ビット・データの転送だけでなく16ビット・データの転送

も行うことができます。これら一群の命令を**16ビット・ロード命令**と総称します。ニーモニック表記上16ビット・ロード命令は2つに分けられます。

16ビット・ロード命令 { LD 命令 / PUSH, POP 命令 }

PUSH と POP は「スタック」という考え方が不可欠なので3.7節以降にまわし，まず LD 命令の方から見ていきます。

LD 命令のニーモニック表記は8ビット・ロード命令と同じく

LD デスティネーション，ソース

の形をしています。ただし，今度は扱うデータが16ビットである点が大きく異なります。

まず「基本形」として次の命令を考えます。

LD HL, 1234H

これは「16ビットデータ 1234H を HL レジスタ・ペアにロードせよ。」と読みます。では，このとき H レジスタ，L レジスタの内容は各々何でしょうか？疑問を感じたら，まず実験です。次のプログラムをご覧ください。

アドレス	マシンコード	ニーモニック
D000	21 34 12	LD HL, 1234H
D003	7C	LD A, H
D004	32 10 D0	LD (0D010H), A
D007	7D	LD A, L
D008	32 11 D0	LD (0D011H), A
D00B	C9	RET

図3.4 16ビット・ロード命令の実験（I）

付録の「命令表」を使ったハンド・アセンブルはできるようになっていますか？ また，アセンブル時の「上下位逆転の原則」はよろしいですか？ 末尾の「オマジナイ」C9も忘れずに！
では，このプログラムをモニター**M**コマンドで書き込み，**G**コマンドで実行してください。

```
MON
*M D000
:D000=21 34 12
:D003=7C
:D004=32 10 D0
:D007=7D
:D008=32 11 D0
:D00B=C9
:D00C=00
*G D000
*■
```

図3.5 実　行（I）

実行結果は，Hレジスタの内容がメモリーのD010H番地に，Lレジスタの内容がD011H番地に格納されるはずですから，ダンプして確認しましょう。

```
*D D000
:D000=21 34 12 7C 32 10 D0 7D /!4.|2.ﾐ}
:D008=32 11 D0 C9 00 00 00 00 /2.ﾐﾉ....
:D010=12 34 00 00 00 00 00 00 /.4......
:D018=00 00 00 00 00 00 00 00 /........
:D020=00 00 00 00 00 00 00 00 /........
:D028=00 00 00 00 00 00 00 00 /........
:D030=00 00 00 00 00 00 00 00 /........
:D038=00 00 00 00 00 00 00 00 /........
:D040=00 00 00 00 00 00 00 00 /........
:D048=00 00 00 00 00 00 00 00 /........
:D050=00 00 00 00 00 00 00 00 /........
:D058=00 00 00 00 00 00 00 00 /........
:D060=00 00 00 00 00 00 00 00 /........
:D068=00 00 00 00 00 00 00 00 /........
:D070=00 00 00 00 00 00 00 00 /........
:D078=00 00 00 00 00 00 00 00 /........
*■
```

図3.6 確　認（I）

予想通り，Hレジスタ＝12H，Lレジスタ＝34Hとロードされていましたね。

3.5 上位バイトと下位バイトの逆転

では，次の例を考えてください。

LD HL,(0D020H)

という命令がありますが，このときH，Lの各レジスタの内容はどうでしょうか。ただし，メモリーのD020H番地のあたりには次のデータがあるとします。(Mコマンドでセットしておいてください。)

```
*D D020 D027
:D020=41 42 43 44 45 46 47 48 /ABCDEFGH
*■
```

図3.7 データのセット

これも実験です。先のプログラムに少し手を加えた次のプログラムを考えます。

アドレス	マシンコード	ニーモニック
D000	2A 20 D0	LD HL, (0D020H)
D003	7C	LD A, H
D004	32 10 D0	LD (0D010H), A
D007	7D	LD A, L
D008	32 11 D0	LD (0D011H), A
D00B	C9	RET

図3.8 16ビット・ロード命令の実験 (II)

では，メモリーに書き込んで実行してみましょう。

```
*M D000
:D000=2A 20 D0
:D003=7C
:D004=32 10 D0
:D007=7D
:D008=32 11 D0
:D00B=C9
:D00C=00
*G D000
*■
```

図3.9

実行結果はどうでしょうか？　おそらく多くの読者の予想に反して，次のようになっているはずです。

```
*D D000
:D000=2A 20 D0 7C 32 10 D0 7D /* ミ|2.ミ}
:D008=32 11 D0 C9 00 00 00 00 /2.ミ)....
:D010=42 41 00 00 00 00 00 00 /BA......
:D018=00 00 00 00 00 00 00 00 /........
:D020=41 42 43 44 45 46 47 48 /ABCDEFGH
:D028=00 00 00 00 00 00 00 00 /........
:D030=00 00 00 00 00 00 00 00 /........
:D038=00 00 00 00 00 00 00 00 /........
:D040=00 00 00 00 00 00 00 00 /........
:D048=00 00 00 00 00 00 00 00 /........
:D050=00 00 00 00 00 00 00 00 /........
:D058=00 00 00 00 00 00 00 00 /........
:D060=00 00 00 00 00 00 00 00 /........
:D068=00 00 00 00 00 00 00 00 /........
:D070=00 00 00 00 00 00 00 00 /........
:D078=00 00 00 00 00 00 00 00 /........
*■
```

図3.10　確　　認（II）

この実験の意味は大切です。今の場合，**LD HL, (0D020H)**により，次のようにデータが転送されたことがわかります。

Hレジスタ ← D020H 番地＝41H
Lレジスタ ← D021H 番地＝42H

すなわち，一般に**LD HL, (mn)**という命令は次の機能をします。

Hレジスタ ↖↗ **mn番地の内容**
Lレジスタ ↙↘ **mn＋1番地の内容**

これが第2の「上下位逆転」です。今度は，アセンブル時ではなく，16ビットのレジスタ（レジスタ・ペアを含む）とメモリー上の2バイトの間でのデータ転送時に生ずる上位バイト，下位バイトの逆転といえます。

次の命令も転送の方向が逆ですが，同様の現象が生じています。

LD（mn），HLの機能

ただし，**LD SP, HL**のようなレジスタ間転送や，**LD IX, mn**のような数値の直接ロードでは「上下位逆転」は生じません。

3.6　スタックとスタック・ポインタ

Z80CPUの特徴の1つは，レジスタが豊富に用意されていて，ユーザーが様々な用途に使い分けられることですが，それでもCPU内のレジスタでは数に限りがありますから，少し複雑なことをしようとすると足りなくなってしまいます。そのようなとき，レジスタ内容を一時的にメモリーに退避させる必要が生じます。これを**LD**命令でユーザーが毎回アドレス管理をして実行していたら，ユーザーの負担は大変ですね。CPUの方で「自動的に」管理してくれたら…とは唯もが思うところです。このような目的のために設けられているのが「スタック」という概念です。

スタック（stack）という言葉を辞書で引くと，「干し草の山，積み重ね」と出ています。コンピュータ用語では，これを「保存したいデータを積み重ねておくためのメモリー上の領域」という意味で用いています。この状況はちょうど私たちが机の上に「ツン読」として本を積み重ねることに似ています。

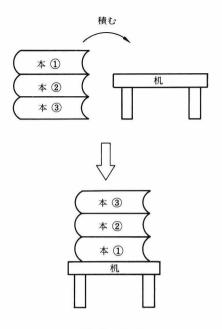



図3.11 ツン読

上図のように本が3冊あったとして，机の上に積み重ねると,「最後に重ねた本が最初に取り出せる」ことになりますね。一番下の本を出すには，上の2冊を取り出してからでなければいけません。当然のことなのですが，コンピュータ用語ではこれを「**Last In First Out**」(**ラスト・イン・ファースト・アウト**＝最後に入れたものが最初に出せるの意味), 略して**LIFO**などとよんでいます。

本の例をメモリーにデータを積み重ねることに置き換えて考えてみましょう。

まず「机」に相当するもの —— どこからデータを積み重ねるかを示すメモリー番地の基準値 —— がなければなりません。そこから「上」にデータを積んでいきますが，メモリーで「上」とは本書では番地の若い方へ向かう方向を指すことにします。

事情は次図のようになります。

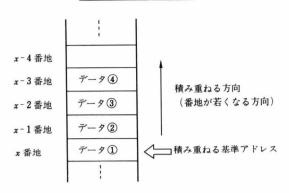



図3.12　メモリーへの積み重ね

これがスタックの様子なのです。スタックに積み重ねられているデータたちの最も「上」の（つまり最も番地の若い）メモリー番地は，**スタック・ポインタ（SP）**とよばれる16ビット・レジスタにより保持されています。

スタックの最初の基準アドレス（「机」の位置にあたる！）を設定するには，

```
LD   SP, mn
```

という16ビット・ロード命令を使います。

この命令が実行されると，スタックがメモリーのmn番地から「上」（番地の若い方）に設定されます。

3.7　PUSH命令

次にスタックへデータを積み重ねる命令を学びましょう。**PUSH**（プッシュ）というOPコードをもつ命令がそれです。**PUSH**命令のオペランドはレジスタ・ペアおよびインデックス・レジスタです。このように**PUSH**命令は，16ビット・ロード命令の一種なので，データの積み重ねはつねに2バイト単位で行われます。

では実験にかかります。次のプログラムをご覧ください。

アドレス	マシンコード	ニーモニック
D000	31 20 D0	LD SP, 0D020H
D003	21 34 12	LD HL, 1234H
D006	E5	PUSH HL
D007	C3 00 00	JP 0000H

図3.13 PUSH 命令の実験

まだ知らない命令が登場しています。**JP 0000H** というニーモニックで表わされ，マシンコード C3 00 00 を持つ 3 バイト命令ですね。これは「ジャンプ命令」の一種で，後で詳しく学びますが，「プログラム・カウンタ（PC）に強制的に 0000H をロードすることにより，0000H 番地へジャンプさせる」命令なのです。なぜこんなことをするかというと，**RET** 命令の C9 では今度のプログラムは停止できないからです。この理由も「サブルーチンからの戻り方」を理解すれば明らかになりますが，今はやはり「オマジナイ」として先へ進みます。

次のように **M** コマンドで，オブジェクトをメモリーに書き込みます。

```
MON
*M D000
:D000=31  20  D0
:D003=21  34  12
:D006=E5
:D007=C3  00  00
:D00A=00
*█
```

図3.14 オブジェクトの書き込み

さて，**＊ G　D000**により実行すると……今度は，画面クリア後 **OK** が表示され，BASIC に戻ってしまいました。

そうなのです！　**JP0000H** は，X1 の HuBASIC では「BASIC に戻ることで

プログラムを停止させる」機能を持つのです。今の場合，スタック・ポインタをプログラム中で，勝手にD020H番地から設定しましたので，このような停止法を採用したわけです。

では，実行結果を見ましょう。

```
Ok
MON
*D D000
:D000=31 20 D0 21 34 12 E5 C3 /1 ミ!4.♣テ
:D008=00 00 00 00 00 00 00 00 /........
:D010=00 00 00 00 00 00 00 00 /........
:D018=00 00 00 00 00 00 34 12 /......4.
:D020=00 00 00 00 00 00 00 00 /........
:D028=00 00 00 00 00 00 00 00 /........
:D030=00 00 00 00 00 00 00 00 /........
:D038=00 00 00 00 00 00 00 00 /........
:D040=00 00 00 00 00 00 00 00 /........
:D048=00 00 00 00 00 00 00 00 /........
:D050=00 00 00 00 00 00 00 00 /........
:D058=00 00 00 00 00 00 00 00 /........
:D060=00 00 00 00 00 00 00 00 /........
:D068=00 00 00 00 00 00 00 00 /........
:D070=00 00 00 00 00 00 00 00 /........
:D078=00 00 00 00 00 00 00 00 /........
*■
```

図3.15　実行結果の確認

Hレジスタの内容(12H)が，D01FH番地に，Lレジスタの内容(34H)が，D01EH番地に積み重ねられていますね。スタックは，D01FH番地より「上」に設定され，その様子は次図のようになります。

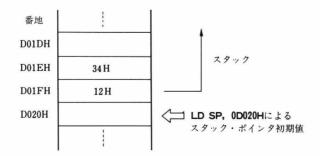



図3.16　スタックの様子

PUSH 命令も16ビット・ロード命令の一種ですから「上位バイトと下位バイトの逆転」が生じ，Hレジスタが「下」にLレジスタが「上」になって積み重ねられることに注意しましょう。

3.8 スタック・ポインタの変化

スタックを利用することの利点は，前節で述べた過程が「自動的」に繰り返せることです。すなわち，PUSH 命令を実行すると，スタック・ポインタ(SP)の内容は自動的に2つだけ減じられ，積み重ねたデータの一番「上」のアドレスをつねに指すように設計されていますから，ユーザーはSPの初期値を設定しておくだけで，以後次々とデータを積み重ねていけます。

たとえば，

```
LD   SP, mn
PUSH HL
PUSH DE
PUSH BC
```

を実行後は，SP の内容は mn-6 番地を示していることになります。

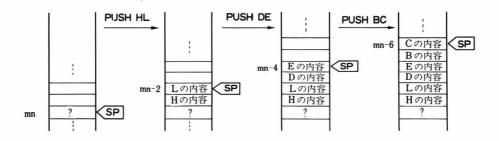



図3.17 PUSH の実行と SP の変化

このように SP 値は，自動的に変更されていくので，これを無制限に行うと大変なことになります。すなわち，他のデータ領域や，プログラム格納領域と衝

突する危険がありますから，スタックの領域として何バイト分が確保されているかだけは注意しておく必要があります。先に **PUSH** 命令の実験で使ったプログラムでは，スタック領域としては，D01FH 番地から「上」に向かって，プログラム終了(C3 00 00)番地の次の D00AH 番地まで，計22バイトが使えることになります。このスタックには，**PUSH** 命令を11回まで使ってデータを積み重ねられますが，これを**スタックの深さ**とよぶことがあります。

3.9 POP 命令

今度は，スタックからデータを取り出す命令を学びます。**POP**（ポップ）命令とよばれるものがそれで，**PUSH** 命令と同じく，レジスタ・ペアまたはインデックス・レジスタをオペランドとします。

たとえば，**POP HL** という命令を実行すると，現在のスタック・ポインタ(**SP**)が示すアドレスとその次のアドレスから2バイトが「上下位逆転」して，HL レジスタ・ペアに転送されます。同時に，SP の内容は自動的に＋2されます。その様子を図示したものが次図です。

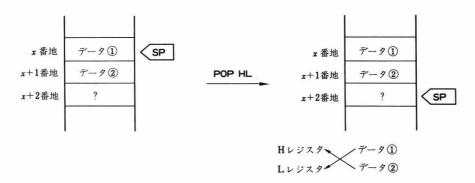



図3.18 POP HL の実行と SP の変化

3.10 PUSH と POP の基本的使い方

スタックへのデータの積み重ねと，取り出しは，3.6節で学んだようにLIFO原則に従いますから，取り出す順番に注意します。**PUSH** 命令と **POP** 命令の基本的な使い方は次のようなものです。

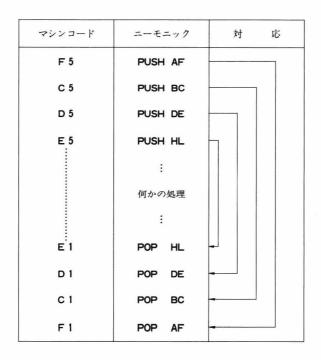



図3.19 PUSH と POP の基本的使い方

このようにLIFO原則の基本に忠実にスタックとのデータのやりとりをすれば，途中の処理でレジスタの内容が変化しても，最終的に元に戻すことができるわけです。そこで，上のような使用法を「レジスタ内容の保護」といいます。

［注］ **POP** 命令も16ビット・ロード命令の一種ですから，「上位バイトと下位バイトの逆転」が生じています。しかし，上のように **PUSH** と **POP** を対で用いると「逆転の逆転」で元に戻るのです。

3.11 ありそうで存在しない命令の実現法

PUSHとPOPのもう少し面白い使い方を紹介します。次の命令列をご覧ください。

```
PUSH   HL
POP    IX
```

これを実行するとどうなりますか？ そう，スタックを経由してHLレジスタの内容がIXレジスタに転送されますね。これは機能としては

```
LD  IX,  HL
```

と同等ですが，Z80 CPUの命令には，こんなLD命令は存在しないのです。

皆さんも，Z80 CPUのマシン語の基礎を勉強されて，自分でプログラムを書き始めるとまもなく気づくことですが，Z80 CPUには上のような「ありそうで存在しない命令」がかなりあります。これはZ80 CPUが，インテル8080 CPUの設計方針を継承していることからくる「不透明」な点なのですが，X1シリーズのメインCPUがZ80Aである以上，これも受け入れなくてはなりません。そこで，「ありそうで存在しない命令」に対しては，同等の働きをする命令列で置き換える便法を使うことになります。上で紹介したのは，そのようなテクニックの1つです。

第4章　マシン語での計算

「電子計算機」というのがコンピュータの訳語です。文字通り，これまでのコンピュータの大切な要素は「大量かつ高速な計算」にあったといえましょう。しかし，最近のコンピュータは「情報処理機械」さらには「知能機械」としての側面が重要になってきているようです。

それはともかく，Z80 CPUのマシン語を操って「計算」をするのが本章の目標です。「計算」といってもZ80には，たし算とひき算の命令しかありません。これらを用いて「算術」をしてみましょう。また，私たちの日常感覚とは少し離れますがZ80には**AND**や**OR**など「論理演算」をする機能があります。その意味で，本章で登場する命令は「演算命令」と総称されます。

4.1　補数と符号つき整数

本章では演算命令に関する説明をいたしますが，まず2進数における「負の整数」の扱いについて考えておきましょう。

私たちが日常的にしている10進数の計算では，桁数制限について特に意識することは少ないでしょう。しかし，コンピュータ内部で行われる2進数の計算においては，扱える桁数に一定の制限があることに注意しておく必要があります。

Z80CPUの場合は，2進数の演算は8ビットあるいは16ビットの形で行われます。

このように桁数制限が存在し，扱える2進数がnビットの形であるような場合には，「負の整数」について独特の考え方をします。正の整数xに対して，その符号を変えた$-x$とは何だったかを反省すると，$x+(-x)=0$という関係が本質であることがわかります。さて今の場合，数の範囲は，$0\sim 2^n-1$までとな

っていることに注意しましょう。2^n は，n ビットで切ってしまうと，0と一致してしまうことに注目して，次のような式を考えます。

$$-x=\{(2^n-1)-x\}+1-2^n$$

これは x が何であっても成り立つ恒等式ですが，以下のように解釈することができます。

① 2^n-1 は n 個のビットがすべて1の2進数である。
② $(2^n-1)-x$ は，x の全ビットの0と1を反転させることに対応する。
③ それに1を加えてから 2^n を無視すれば $-x$ に等しくなる。

すなわち，$-x$ の作り方を表していると考えられます。①②③のような手順で作られる $-x$ を x に対する「**2の補数**」あるいは単に「**補数**」といいます。

2の補数の考え方

2進数 x に対して，$-x$ を作るには，x の全ビットの1と0を反転してから，1を加えればよい．

図4.1

たとえば，8ビットの2進数を扱うときには，$x=06\mathrm{H}$ に対する補数は次のようになります。

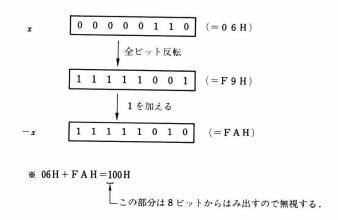
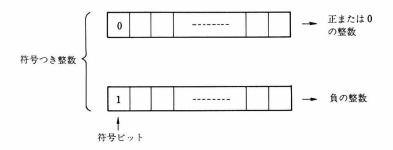



図4.2 補数の例

このように補数の考え方をすると，負の整数も扱えます。これを**符号つき整数**とよびます。符号つき整数では，最も左端のビット位置が**符号ビット**と解釈されます。

したがって，n ビットの符号つき整数では，扱える範囲は-2^{n-1}～$+2^{n-1}-1$ということになります。たとえば，8ビットの場合は，符号なし整数としては0～255が考えられますが，符号つき整数では－128～＋127の範囲で考えます。

これらの関係は次の通りです。

2進表示	符号なし整数(10進)	符号つき整数(10進)	16進表示
1 1 1 1 1 1 1 1	255	−1	F F H
1 1 1 1 1 1 1 0	254	−2	F E H
1 1 1 1 1 1 0 1	253	−3	F D H
⋮	⋮	⋮	⋮
1 0 0 0 0 0 0 0	128	−128	80H
0 1 1 1 1 1 1 1	127	+127	7 F H
0 1 1 1 1 1 1 0	126	+126	7 E H
⋮	⋮	⋮	⋮
0 0 0 0 0 0 1 0	2	+2	02H
0 0 0 0 0 0 0 1	1	+1	01H
0 0 0 0 0 0 0 0	0	0	00H

図4.3 符号なし整数と符号つき整数（8ビット）

このように1つの2進整数は「符号なし」と「符号つき」の2つの解釈を持つことに注意しましょう。

符号つき整数と16進数の一覧表を付録3に掲げておきましたので，利用してください。

4.2 フラグ

BASICプログラムにおいても，何か意味のある処理をしようとすると**IF〜THEN**を用いた条件判断を避けることはできません。この事情はマシン語プログラムにおいても全く同様で，Fレジスタを参照することにより行われます。

Fレジスタは**フラグ・レジスタ**ともよばれる8ビット・レジスタです。このレジスタは，アキュムレータや汎用レジスタたちと異なって，CPUの働きに応じて自動的に内容が書きかえられてしまう点に特徴があります。Fレジスタの各

ビットは**フラグ**(flag＝旗)とよばれます。8本の旗が横一列に並んでいる様子を思い浮かべましょう。

旗が立っているのは，そのビットが1であることを示し，旗が下りているのはビットが0であることを意味しています。

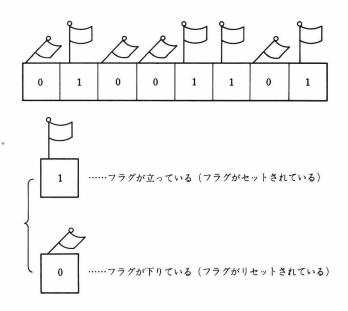



図4.4 Fレジスタの様子

これら8個の**フラグ・ビット**は，CPUの動作により，一定の規則に従ってパタパタと頻繁にセット，リセットされていきます。

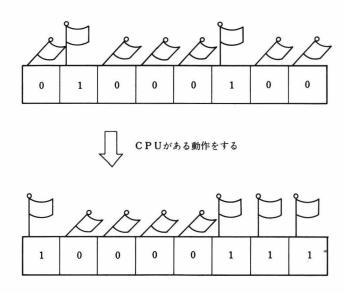



図4.5　フラグのセット，リセット

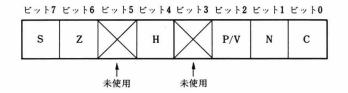
フラグ・ビットのうち6個には，各々特別な意味があり，働きに応じた名称がつけられています（残り2個のビットは使われていません)。

S ……サイン・フラグ
Z ……ゼロ・フラグ
H ……ハーフ・キャリー・フラグ
P/V……パリティー/オーバー・フロー・フラグ
N ……加/減算フラグ
C ……キャリー・フラグ(CyあるいはCYとも書く)

図4.6　フラグ・ビットの名称

4.3 フラグ変化の規則

各フラグの変化する規則は，かなり複雑ですから，詳細は演算命令の説明中および付録2の命令表を見ていただくことにして，ここでは大体の働きを説明しておきます。

キャリー・フラグ(Cy フラグ)は，加算命令で最上位ビットからの桁上げ(キャリー）があったとき，減算命令で小から大を減じたときの桁借り（ボロー)があったときなどにセットされます。

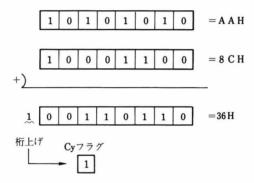
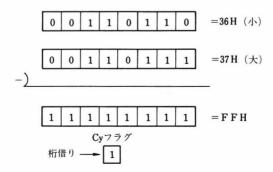



図4.7 キャリー・フラグがセットされる例

加/減算フラグ（N フラグ）は，加算命令でリセットされ，減算命令でセットされます。

パリティー/オーバー・フロー・フラグ（P/V フラグ）は，2つの機能を持ちます。まず「符号つき整数」と見なしたときの算術演算結果が，−128～＋127の範囲を越えていると，P/V フラグは**オーバー・フロー（V）**としてセットされます。また論理演算やビット回転を行ったときには，結果の**パリティ**（1バイト中の〝1〞ビット個数の偶奇）により，**P＝0（パリティ奇数）**あるいは**P＝1（パリティ偶数）**となります。

ハーフ・キャリー・フラグ（H フラグ）は，8ビットの算術演算でのビット3とビット4の間でのキャリー，ボローの有無により，セットあるいはリセットされます。

ゼロ・フラグ（Z フラグ）は，命令の実行結果がゼロならセットされ，そうでなければリセットされます。キャリー・フラグと並んで重要なフラグです。

サイン・フラグ（S フラグ）は，演算実行結果を「符号つき1バイト」と見たときの符号ビット（ビット7）の内容を保持しています。「サイン」は sign のことで「符号」を意味する言葉です。

以上，各フラグの基本的変化だけでも覚えるのは大変ですね。ここでは最低限キャリー・フラグとゼロ・フラグだけでも理解しておいてください。

①　減算　12H－34Hを実行したとき

Fレジスタ	?	0	?	?	?	?	?	1
		Z						Cy

②　減算　12H－12Hを実行したとき

Fレジスタ	?	1	?	?	?	?	?	0
		Z						Cy

③　減算　34H－12Hを実行したとき

図4.8　Cフラグと Z フラグの変化例

4.4 マシン語での条件判断

BASICの命令にもIF～THENのように条件判断により処理の流れを変えるものがありますが，フラグはマシン語での条件判断に用いられます。

マシン語の命令の中には，実行直前のFレジスタの特定ビットを参照して，それが1か0かにより処理を変えるものがあります。これを**条件つきの命令**とよびます。たとえば，JP Z, mnというニーモニックで表記される命令は条件つきジャンプ命令の1つで，「Zフラグが1ならば，mn番地へジャンプせよ。そうでなければ，何もせず次の命令へ進め。」という指示をします。

ここで，ニーモニック内に書かれる条件判断表記とその意味をまとめておきましょう。

条件判断	意　　味
C	キャリー・フラグが1であれば,
NC	キャリー・フラグが0であれば,
Z	ゼロ・フラグが1であれば,
NZ	ゼロ・フラグが0であれば,
PE	パリティ偶数（P/Vフラグ=1）であれば,
PO	パリティ奇数（P/Vフラグ=0）であれば,
M	サイン・フラグが1であれば,
P	サイン・フラグが0であれば,

図4.9 条件判断の表記法

4.5 8ビット算術演算命令

では，いよいよZ80 CPUの演算命令について学びましょう。演算命令には8ビット対象のもの，16ビット対象のものがあります。また，演算の種類から分けると算術演算，論理演算に区別されます。

まず，8ビット算術演算命令から見ましょう。**算術演算**というのは加算（たし算）や減算（ひき算）に関する演算です。Z80では乗算（かけ算）・除算（わり算）の命令は設けられていませんから，これらは加算・減算の組み合わせで実現する必要があります。

4.6　加算命令

加算命令には，ADD命令とADC命令があります。これらはいずれも2個のオペランドを持ち，第1オペランドは必ずAレジスタと決められています。たとえば，

ADD　A，B

というニーモニックで表される加算命令では，Aレジスタの内容とBレジスタの内容を加算し，結果がAレジスタに格納されます。

① Aレジスタ=44H
Bレジスタ=11H　}のとき **ADD A，B**
を実行すると，
Aレジスタ=55H
Bレジスタ=11H　}になります．

② Aレジスタ=78H
Bレジスタ=69H　}のとき **ADD A，B**
を実行すると，
Aレジスタ=E1H
Bレジスタ=69H　}になります．
このときSフラグ=1，P/Vフラグ=1となります．

③ Aレジスタ=FBH
Bレジスタ=F1H　}のとき **ADD A，B**
を実行すると，
Aレジスタ=ECH
Bレジスタ=F1H　}になります．
このときSフラグ=1，Cyフラグ=1となります．
今度は，符号つき整数としてのオーバー・フローは発生していませんから，P/Vフラグ=0です．

図4.10　ADD　A，Bの例

次にADD命令の実験プログラムを掲げます。

アドレス	マシンコード	ニーモニック
D000	31 30 D0	LD SP, 0D030H
D003	3E FB	LD A, 0FBH
D005	06 F1	LD B, 0F1H
D007	80	ADD A, B
D008	F5	PUSH AF
D009	C3 00 00	JP 0000H

図4.11 ADD命令の実験

```
MON
*M D000
:D000=31 30 D0
:D003=3E FB
:D005=06 F1
:D007=80
:D008=F5
:D009=C3 00 00
:D00C=00
*■
```

図4.12 オブジェクトの書き込み

Aレジスタと Fレジスタの内容を見るため，ここではスタックを利用しています。

PUSH AFにより，実行後のAレジスタの内容がD02FH番地に，またFレジスタの内容がD02EH番地に格納されます。**＊G D000**により実行すると，BASICのコマンド・レベルに戻って停止しますから，再びモニターを起動し，**＊D D000**によりメモリーをダンプして結果を確認しましょう。

```
Ok
MON
*D D000
:D000=31 30 D0 3E FB 06 F1 80 /10ミ>年.土_
:D008=F5 C3 00 00 00 00 00 00 /火テ......
:D010=00 00 00 00 00 00 00 00 /........
:D018=00 00 00 00 00 00 00 00 /........
:D020=00 00 00 00 00 00 00 00 /........
:D028=00 00 00 00 00 00 A9 EC /......ウ■
:D030=00 00 00 00 00 00 00 00 /........
:D038=00 00 00 00 00 00 00 00 /........
:D040=00 00 00 00 00 00 00 00 /........
:D048=00 00 00 00 00 00 00 00 /........
:D050=00 00 00 00 00 00 00 00 /........
:D058=00 00 00 00 00 00 00 00 /........
:D060=00 00 00 00 00 00 00 00 /........
:D068=00 00 00 00 00 00 00 00 /........
:D070=00 00 00 00 00 00 00 00 /........
:D078=00 00 00 00 00 00 00 00 /........
*■
```

図4.13 実行結果の確認

上の例では，A レジスタの内容は ECH に変化し，F レジスタは A9H ですから各フラグ・ビットは次のようになっているわけですね。

S	Z	×	H	×	P/V	N	Cy
1	0	1	0	1	0	0	1

D004H 番地と， D006H 番地の内容をいろいろに変えて，皆さんで実験を試み， 8 ビット加算とフラグ変化について理解を深めてください。

ADC 命令は，**ADD** 命令とほとんど同じですが，この命令の実行直前の Cy フラグの値(1 か 0)がさらに加えられる点が異なります。たとえば， **ADC A, B** では， A レジスタの内容と B レジスタの内容および Cy フラグの値が加算され，結果は A レジスタに格納されます。

4.7 減算命令

減算命令には，**SUB** 命令と **SBC** 命令があります。ただし，今度はニーモニック表記で注意する点があります。いずれも演算される側のレジスタがAレジスタであるのは共通なのですが，**SBC** 命令では加算命令と同様に2個のオペランドを指定し，第1オペランドはAレジスタなのに対し，**SUB** 命令ではオペランドは1個で，Aレジスタに対し演算する側のレジスタ名のみ書きます。たとえば，**SUB D** は正しい表記ですが，**SUB A, D** は誤りです。

次のプログラム例をご覧ください。

アドレス	マシンコード	ニーモニック
D000	31 30 D0	LD SP, 0D030H
D003	3E 05	LD A, 5
D005	16 07	LD D, 7
D007	92	SUB D
D008	F5	PUSH AF
D009	C3 00 00	JP 0000H

図4.14 SUB 命令の実験

これは，5－7を計算するプログラムです。CPU 内部では，5＋(－7)の形で補数を用いた加算として扱われます。結果は「符号つき整数」としてAレジスタに格納されます。では，オブジェクトを書き込み，実行してみましょう。

```
MON
*M D000
:D000=31 30 D0
:D003=3E 05
:D005=16 07
:D007=92
:D008=F5
:D009=C3 00 00
:D00C=00
*G D000
```

図4.15 オブジェクトの書き込み

```
Ok
MON
*D D000
:D000=31 30 D0 3E 05 16 07 92 /10ﾐ>...ｰ
:D008=F5 C3 00 00 00 00 00 00 /火ﾃ......
:D010=00 00 00 00 00 00 00 00 /........
:D018=00 00 00 00 00 00 00 00 /........
:D020=00 00 00 00 00 00 00 00 /........
:D028=00 00 00 00 00 00 BB FE /......ｻ生
:D030=00 00 00 00 00 00 00 00 /........
:D038=00 00 00 00 00 00 00 00 /........
:D040=00 00 00 00 00 00 00 00 /........
:D048=00 00 00 00 00 00 00 00 /........
:D050=00 00 00 00 00 00 00 00 /........
:D058=00 00 00 00 00 00 00 00 /........
:D060=00 00 00 00 00 00 00 00 /........
:D068=00 00 00 00 00 00 00 00 /........
:D070=00 00 00 00 00 00 00 00 /........
:D078=00 00 00 00 00 00 00 00 /........
*█
```

図4.16 結果の確認

いかがですか？ Aレジスタの内容は－2に対応する「1バイト符号つき整数」FEHになっていますね。このときのFレジスタの内容はBBHですから，各フラグは次のようになっています。

S	Z	×	H	×	P/V	N	Cy
1	0	1	1	1	0	1	1

小から大を減じて桁借り（ボロー）がありますからCy＝1，結果は負ですからS＝1，減算をしましたからN＝1となっています。

SBC命令では，実行直前のCyフラグの内容がさらに減じられます。たとえば，**SBC A, D**では，Aレジスタの内容からDレジスタの内容を減じ，さらに直前のCyフラグの値（1か0）を減じて，その結果がAレジスタに格納されます。

4.8 比較命令

減算の特殊な形として，比較命令があります。比較命令ではAレジスタとオペランドに指定されたレジスタ，数値との大小が比較されます。ニーモニックではOPコードは**CP**，オペランドは**SUB**命令と同じく1個のみ持ちます。たとえば，**CP D**では，Aレジスタの内容からDレジスタの内容を減ずることでフラグを変化させ，大小の判断をします。ただし，減算命令との決定的違いは減算の答は出力されず，Aレジスタの内容は**CP**命令実行前のまま保持される点です。次のプログラム例を実行して，**SUB D**との違いを理解してください。

アドレス	マシンコード	ニーモニック
D000	31 30 D0	LD SP, 0D030H
D003	3E 05	LD A, 5
D005	16 07	LD D, 7
D007	BA	CP D
D008	F5	PUSH AF
D009	C3 00 00	JP 0000H

図4.17 CP命令の実験

```
Ok
MON
*D D000
:D000=31 30 D0 3E 05 16 07 BA /10ミ>...コ
:D008=F5 C3 00 00 00 00 00 00 /炎テ......
:D010=00 00 00 00 00 00 00 00 /........
:D018=00 00 00 00 00 00 00 00 /........
:D020=00 00 00 00 00 00 00 00 /........
:D028=00 00 00 00 00 00 93 05 /......T.
:D030=00 00 00 00 00 00 00 00 /........
:D038=00 00 00 00 00 00 00 00 /........
:D040=00 00 00 00 00 00 00 00 /........
:D048=00 00 00 00 00 00 00 00 /........
:D050=00 00 00 00 00 00 00 00 /........
:D058=00 00 00 00 00 00 00 00 /........
:D060=00 00 00 00 00 00 00 00 /........
:D068=00 00 00 00 00 00 00 00 /........
:D070=00 00 00 00 00 00 00 00 /........
:D078=00 00 00 00 00 00 00 00 /........
*■
```

図4.18 結果の確認

この例では，Ｚフラグ＝０であることからＡレジスタ≠Ｄレジスタが，また Cy フラグ＝１からＡレジスタ＜Ｄレジスタがわかりますね。

4.9　増減命令

レジスタやメモリーの内容を，１だけ増やす(**インクリメント**)，あるいは１だけ減らす（**デクリメント**）という処理はカウンタ増減などで頻繁に用いられます。そのために，これらは増減命令 **INC** あるいは **DEC** として特別に用意されています。たとえば，**INC H** ではＨレジスタの内容が＋１され，**DEC L** ではＬレジスタの内容が－１されます。ただし，**ADD** や **SUB** と決定的に違うのは，Cy フラグが変化しない点です。次の例をご覧ください。

```
LD   B, 0FFH
INC  B
```

この例では，FFH＋１により桁上げが起こっていますが，Cy フラグは実行直前のままです。

```
LD   B, 0
DEC  B
```

でも同様で，０－１により桁借りを生じますが，Cyフラグは変化しません。

4.10　8ビット論理演算命令

Z80 の論理計算には **AND**，**OR**，**XOR** の３種類があります。これらを理解するため，まず１ビット単位で考えてみましょう。以下で，ｐやｑは１ビットの値（１か０）を持つものとします。

ｐとｑに対して，次の３つの演算を考えます。

pとqの論理積（and）p∧q

p	q	p ∧ q
0	0	0
0	1	0
1	0	0
1	1	1

pとqの論理和（or）p∨q

p	q	p ∨ q
0	0	0
0	1	1
1	0	1
1	1	1

pとqの排他的論理和
（exclusive or）p⊕q

p	q	p⊕q
0	0	0
0	1	1
1	0	1
1	1	0

図4.19　ビット単位の論理演算

これらの起源は記号論理学に発しているのですが，論理学を知らなくても，上の表で形式的に「定義」された演算であると理解すれば，ここでの目的には十分です。

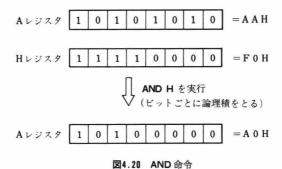
さて，**AND**，**OR**，**XOR** の説明に戻ります。これらの命令のニーモニック表記は1個のオペランドを持ち，オペランドの内容とAレジスタとのビットごとの論理演算（∧，∨，⊕）を行なった後，結果をAレジスタに格納するという機能を持ちます。**SUB** 命令と同様に，演算される側のAレジスタはニーモニック上では表記されません。

4.11　AND 命令と OR 命令

例を見ましょう。

AND　H

という命令は，A レジスタ，H レジスタの初期値に対して次のように作用します。

図4.20　AND 命令

H レジスタには，上位 4 ビットがすべて 1，下位 4 ビットがすべて 0 というデータがありますが，これと A レジスタとのビットごとに論理積をとると，A レジスタの下位 4 ビットが強制的に 0 にされます。ちょうど下位 4 ビットに**マスク**（覆面）がかけられて見えなくなったように考えられますね。A レジスタの特定のビット位置にマスクをかけるには，そのビットを 0 にしたデータ（**マスク・データ**）を用意し，A レジスタとの論理積をとればよいことがわかります。今の例では，H レジスタのデータ F0H がマスク・データになっています。このように，**マスクをかける**のが AND 命令の基本的な使われ方といえます。

逆に，特定のビット位置を強制的に 1 にするには，OR 命令が使われます。たとえば，

OR　0FH

という命令を実行すると，A レジスタの内容は次のように変化します。

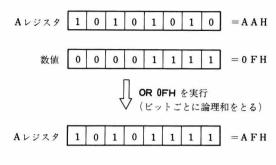
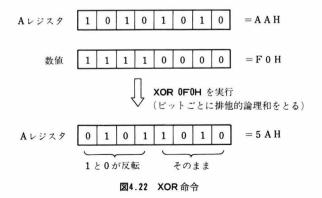



図4.21 OR命令

この例では，数値データ0FHとの論理和をとることで，Aレジスタの下位4ビットを一斉に1にしていますね。

4.12 XOR命令

XOR命令は，Aレジスタとオペランドの間で，ビットごとの排他的論理和(eXclusive OR)をとり，結果をAレジスタに格納するものです。この命令には面白い機能があります。$p \oplus q$の演算表をもう一度ご覧ください。$q=1$のときを見ると，pと$p \oplus q$の値は0と1が反転しています。そうです！ 排他的論理和は「ビット反転」に使えるのです。たとえば，XOR 0F0Hという命令を考えます。

図4.22 XOR命令

この例では，Aレジスタの上位4ビットの1と0を反転しています。Aレジスタのビット・パターンが「ある端子のON/OFF」を表しているとすると，このような機能はON ↔ OFFを行なう**スイッチ機能**と考えることができますね。

それから，p＝qのときp⊕q＝0となることを利用すると，次のような使い方もできます。

XOR　A

この命令を実行すると，Aレジスタの内容はどうなっているでしょうか？そうですね。AレジスタとAレジスタの排他的論理和ですから，Aレジスタの全ビットが0になってしまいます。すなわち，「Aレジスタをクリアする」命令と考えられますね。**LD A,00H**がマシンコードで3EH 00Hと2バイトなのに対し，**XOR A**のマシンコードはAFHと1バイトで済みます。**XOR A**は，非常によく使われますから，是非覚えてください。

4.13　論理演算とフラグ変化

以上，3種類の論理演算を見てきましたが，最後にフラグ変化について注意しておきます。共通の特徴として，論理演算を行うと，必ずCyフラグがリセット（Cy＝0）されます。これに注目すると，次の命令の意味がわかります。

AND　A　あるいは　OR　A

これらの命令は，Aレジスタどうしの**AND**や**OR**ですからAレジスタの内容は変わりません。では何が変化するかというと，フラグが変化するのです。たとえば，Cy＝0となりますから，これらは「Aレジスタの内容を変えずにCyフラグをリセットする命令」として利用されます。また，Aレジスタの内容が00Hであるときには，Zフラグがセット（Z＝1）されますから，「Aレジスタが0かどうかを調べる命令」としても使われます。

Ｚフラグに関しては，次のような使われ方も覚えておくとよいでしょう。

```
LD  A, H
OR  L
```

この２つの命令を相ついで実行すると，Ｚフラグの変化はどうなるでしょうか？　答は，Ｈレジスタ，Ｌレジスタともに０のときのみＺ＝１，他はＺ＝０となります。これは，HLレジスタ・ペアが０かどうかを調べるのに利用されています。

論理演算というと，算術演算に比べてなじみが薄いかもしれません。ある意味でコンピュータ独特の演算といえるでしょう。私もマシン語学習の初期に，こうしたプログラミング・テクニックを見て「手品」のように感じたものでした。皆さんはどのように思われるでしょうか？

4.14　アキュムレータ・フラグ制御命令

本節で扱う命令は算術命令の一種ですが，加・減算とは異なる一群をなすため標題のように別にして扱うことが多いようです。ニーモニック表記上では，いずれもOPコードのみでオペランドは持ちません。

CPL 命令：補数を意味する英語 complement の省略形がニーモニックになっていて，Ａレジスタの内容の全ビットの０，１を反転する命令です。これを「１の補数をとる」と呼ぶ習慣があるため，このような名称がつけられています。

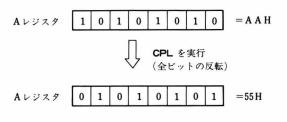



図4.23　CPL 命令

NEG 命令：Aレジスタの内容の「2の補数」をとる命令です。「符号つき整数」x に対して，$-x$ をつくることから名称（negate の略）がつけられています。マシンコードは EDH，44H の2バイトです。

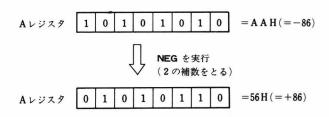



図4.24　NEG 命令

CCF 命令：　complement carry flag の略で，Cy フラグの0，1を反転させます。

SCF 命令：　set carry flag の略で，Cy フラグを1にする（セットする）命令です。

注意すべきは，Z80 の命令セットの中には，Cy フラグを0にする（リセットする）ためだけの命令はありません。したがって，他の命令で代行します。すぐ考えつくのは，上の2つの命令の組み合わせです。

SCF　（Cy＝1にする）
CCF　（Cy を反転する）

をひきつづいて実行すれば，Cy＝0にできます。しかし，もう皆さんはこれを一発でできますね。そうです。論理演算が利用できましたね。**OR A** または，**AND A** を実行すれば，Cy＝0にできるのです。

4.15　2進化10進数と DAA 命令

本節では，**DAA** というニーモニック（オペランドなし）で表される命令を説明します。OP コードは decimal adjust accumulator の省略形を意味し，「10

進補正命令」とよばれています。この命令を理解するには「2進化10進数」について知る必要があります。

2進化10進数（BCD＝binary coded decimal）とは，10進数の数字の並びをそのまま2進数として実現したものです。10進数の数字は，0から9まで10個ですから，4ビット($2^4=16$)あれば表わせますね。たとえば，2＝0010，8＝1000というように。こうして，10進数28を00101000という2進数に対応させることにしたのがBCDです(00101000を普通の2進数と解釈すると，10進数の40に対応しています)。コンピュータ内部の演算はすべて2進数として処理されていますが，2進数としての桁の切り捨て，切り上げと10進数としてのそれとが少し食い違うので，特に小数点を含む計算などで思わぬ誤差を生ずることがあります。このようなときBCDを使えば10進数の感覚のままで計算ができるわけです。しかしながら，CPU内部の計算はあくまで2進数として行われるので，出てきた結果を10進数として解釈するため補正しなくてはなりません。たとえば，BCDとしての16と28のたし算は，BCDとして44になるはずですが，CPU内部では16H＋28H＝3EHとなってしまいます。この場合，3EHを44Hに補正するのが10進補正であり，それを行う命令がDAAであるわけです。ですから，DAA命令は，ふつう加減算の直後に実行します。

次のプログラム例をご覧ください。

アドレス	マシンコード	ニーモニック
D000	21　20　D0	LD HL，0D020H
D003	7E	LD A，(HL)
D004	23	INC HL
D005	86	ADD A，(HL)
D006	27	DAA
D007	23	INC HL
D008	77	LD (HL)，A
D009	C9	RET

アドレス	データ（BCD）
D020	16
D021	28

（注） INC HL
は，HLレジスタ・ペアの内容を＋1する命令.
4.17節を参照.

図4.25　DAA命令

このプログラムは，D020H番地のBCDとD021H番地のBCDを加算して，結果をBCDとしてD022H番地に格納するものです。16＋28＝44となることを確認してみましょう。

```
MON
*M D000
:D000=21  20 D0
:D003=7E
:D004=23
:D005=86
:D006=27
:D007=23
:D008=77
:D009=C9
:D00A=00
*M D020
:D020=16
:D021=28
:D022=00
*■
```

図4.26　オブジェクトのセット

```
*G D000
*D D000
:D000=21  20  D0  7E  23  86  27  23  /! ﾐ‾#■'#
:D008=77  C9  00  00  00  00  00  00  /ω).......
:D010=00  00  00  00  00  00  00  00  /........
:D018=00  00  00  00  00  00  00  00  /........
:D020=16  28  44  00  00  00  00  00  /.(D.....
:D028=00  00  00  00  00  00  00  00  /........
:D030=00  00  00  00  00  00  00  00  /........
:D038=00  00  00  00  00  00  00  00  /........
:D040=00  00  00  00  00  00  00  00  /........
:D048=00  00  00  00  00  00  00  00  /........
:D050=00  00  00  00  00  00  00  00  /........
:D058=00  00  00  00  00  00  00  00  /........
:D060=00  00  00  00  00  00  00  00  /........
:D068=00  00  00  00  00  00  00  00  /........
:D070=00  00  00  00  00  00  00  00  /........
:D078=00  00  00  00  00  00  00  00  /........
*■
```

図4.27　実行と確認

4.16　16ビット算術演算命令

今度は16ビットの演算を考えましょう。対象とするのは，たとえば3456H＋13AFH＝4805Hのような演算です。もちろん，上位バイトと下位バイトに分けて8ビット演算を2回行ない，桁上がり等を考慮すれば実現できますが，Z80には16ビットの加減算をする命令が用意されていますから，これを使います。

8ビットの加減算では，アキュムレータ（Aレジスタ）が中心的役割をはたしましたが，16ビット加減算では主としてHLレジスタ・ペアがアキュムレータの役割をします。ニーモニック表記は，8ビットの場合と同様なので以下に例で説明するにとどめますが，大きな違いは減算命令がSBCというCyフラグ込みのものしか設けられてない点です。SUB命令にあたる命令はないので，OR AなどでCy＝0にしてからSBC命令を実行することで代用します。

アドレス	マシンコード	ニーモニック
D000	21 56 34	LD HL, 3456H
D003	11 AF 13	LD DE, 13AFH
D006	19	ADD HL, DE
D007	22 20 D0	LD (0D020H), HL
D00A	C9	RET

図4.28　16ビット加算命令の例

(3456H＋13AFH＝4805Hを行なう)

上のプログラムを実行すると，D020H番地，D021H番地にHLレジスタの内容（計算結果）が上下位逆転して格納されます。

```
MON
*M D000
:D000=21 56 34
:D003=11 AF 13
:D006=19
:D007=22 20 D0
:D00A=C9
:D00B=00
*G D000
*D D020 D020
:D020=05 48 00 00 00 00 00 00 /.H......
*■
```

図4.29　実行と確認

今度は，4805H－3456H＝13AFHを計算してみましょう。Cy＝0をするのを忘れずに。プログラムと実行例は以下の通りです。

アドレス	マシンコード	ニーモニック
D000	21 05 48	LD HL, 4805H
D003	11 56 34	LD DE, 3456H
D006	B7	OR A
D007	ED 52	SBC HL, DE
D009	22 20 D0	LD (0D020H), HL
D00C	C9	RET

図4.30　16ビット減算命令の例

(4805H－3456H＝13AFHを行なう)

```
MON
*M D000
:D000=21 05 48
:D003=11 56 34
:D006=B7
:D007=ED 52
:D009=22 20 D0
:D00C=C9
:D00D=00
*G D000
*D D020 D020
:D020=AF 13 00 00 00 00 00 00 /ｯ.......
*■
```

図4.31　実行と確認

16ビット加算命令 ADD に限って，インデックス・レジスタを16ビットのアキュムレータとして用いることができます。たとえば，ADD IX, BC や ADD IY, DE などの命令があります。

4.17 16ビット増減命令

これらのほかに，16ビットの増減命令として INC や DEC があります。ここでは，INC HL という命令の使い方を例に説明します。

たとえば，あるものの個数や回数を数えるときに，HL レジスタ・ペアを使うならば，1回カウントするごとに HL の内容を＋1すればよいですね。このようなとき，INC HL を使います。この場合，HL レジスタ・ペアは「16ビットの**カウンタ**として使われている」といいます。

また，HL レジスタ・ペアをメモリー番地を指すのに使うこともありますね。たとえば，LD A, (HL) は HL レジスタ・ペアの示すメモリー番地の内容を A レジスタにロードする命令ですが，ここでの HL レジスタ・ペアの使われ方がそれです。このようなとき，HL レジスタ・ペアは「**ポインタ**として使われている」といいます。メモリー上にデータが連続して格納されているときポインタを先頭番地にセットして，処理するごとにポインタを次に移すことをしますが，HL レジスタ・ペアをポインタに使っていれば，INC HL が「ポインタを＋1する」を実現してくれます。4.15節の図4.25を参考にしてください。

INC 命令の使い方，わかりましたか？　DEC 命令も同様です。

ただし，ここで重要な注意をしておかねばなりません。16ビットの INC, DEC 命令を実行しても，フラグは一切変化しないのです。たとえば，HL＝0001H であったとして，DEC HL を実行しても，Z フラグは変化しません。HL レジスタ・ペアが0000H になったか否かは，別にチェックする必要があって，ふつう

```
LD  A, H
OR  L
```

という命令列が使われます(ただし，Aレジスタの内容がこわれるので必要なら退避しておく)。たとえば，HLをカウンタとして用いる繰り返し処理の流れ図は次のようになります。

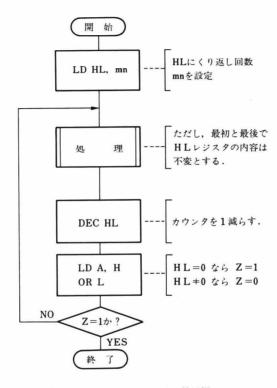



図4.32 16ビット・カウンタの使用例

第5章　プログラムの流れをかえる

電源をONにした直後，Z80 CPUのプログラム・カウンタ（PC）の内容は0000Hに初期化され，CPUは0000H番地から命令を実行していきます。命令を読み込むごとにPCの内容は自動的に＋1ずつされていくので，ふつうは番地の若い方から順に命令が実行されていくのですが，少し複雑なことをしようとすると必ず「条件判断」や「場合分け」などが生じて，上述のようなプログラム実行の流れでは対応しきれなくなります。

Z80の命令の中には，PCの内容を強制的に変更して別の番地に制御を移すものがいくつか設けられています。

本章では，こうした命令を扱います。また，マシン語プログラムにおけるサブルーチンの考え方についても詳述します。

5.1　ジャンプ命令

今まで登場してきたプログラム中で，**JP 0000H**という命令が登場してきましたが，本節ではこの種の「ジャンプ命令」を解説します。

この命令はBASIC言語での**GOTO**文に相当するもので，プログラム実行の流れを変更するためのものです。ただし，BASICの**GOTO**文では「行番号」を飛び先として指定するのに対して，ジャンプ命令では「メモリー番地」を指定します。

ジャンプ命令には，**JP**（＝jump）というOPコードを持つ「絶対ジャンプ」と，**JR**（＝jump relative）というOPコードを持つ「相対ジャンプ」があります。

5.2 絶対ジャンプ（無条件）

絶対ジャンプ命令は，飛び出し先のメモリー番地 mn（絶対番地）を直接に指定するもので，

無条件ジャンプ **JP mn**
条件付ジャンプ **JP cc, mn**

の2系列に分れます。

JP mn は，プログラム・カウンタ（PC）に2バイト値 mn をロードする命令です。したがって，この命令を実行すると，CPU は無条件で mn 番地に制御を移します。オブジェクトは C3H，n，m からなる3バイト命令です。2バイト mn のアセンブルで上下位逆転が起こりますから注意しましょう。m は飛び先番地の上位バイト，n は下位バイトです。今まで何度か登場した **JP 0000H** という命令は，ですから「0000H 番地へジャンプせよ」の命令になりますね。では，0000H 番地へジャンプした後はどうなるのでしょう。モニターを起動し，0000H 番地からをダンプしてください（図5.1）。

「上下位逆転」について補足

60ページに登場した「上下位逆転」について説明の補足をします。2バイトデータの転送時の「上下位逆転」は，番地の数字の若い方を「上」としたため生じる見かけの現象に過ぎません。モニターでダンプしたとき，メモリー番地の若い方が画面で「上」になることから，本書では敢えてこうした見方をしてみました。わかりにくいと思われる方は，発想を逆転して番地の大きい方を「上」だと考えてください。60ページの説明も無理なくご理解いただけることでしょう。

```
X1 ｼﾘｰｽﾞ ﾉ ﾊﾞｲﾄ

:0000=C3 FA 00 C3 7C 01 50 28 /ﾃ..ﾃ|.P(
:0008=C3 D3 03 C3 83 04 00 18 /ﾃﾓ.ﾃ▄...
:0010=C3 D3 03 C3 BC 04 00 18 /ﾃﾓ.ﾃｼ...
:0018=C3 D3 03 C3 9D 02 00 27 /ﾃﾓ.ﾃ..'
:0020=C3 D3 03 C3 07 0E 07 20 /ﾃﾓ.ﾃ...
:0028=C3 D3 03 C3 AA 0A 00 F7 /ﾃﾓ.ﾃｪ..日
:0030=C3 D3 03 C3 CA 0A 00 00 /ﾃﾓ.ﾃﾊ...
:0038=C3 D3 03 C3 75 0B C3 79 /ﾃﾓ.ﾃu.ﾃy
:0040=0B C3 9A 0B C3 9E 0B C3 /.ﾃ.ﾃ.ﾃ
:0048=AE 0B C3 30 03 C3 88 09 /ｮ.ﾃ0.ﾃ.
:0050=00 00 46 03 D3 03 D3 03 /..F.ﾓ.ﾓ.
:0058=D3 03 D3 03 D3 03 D3 03 /ﾓ.ﾓ.ﾓ.ﾓ.
:0060=D3 03 D3 03 D3 03 C3 FA /ﾓ.ﾓ.ﾓ.ﾃ.
:0068=00 63 0E 63 0E 8B 07 1E /.c.c.▮..
:0070=07 63 0E F1 07 A8 07 F7 /.c.土.ｨ.日
:0078=07 14 08 A1 08 1B 07 BF /...｡...ｿ
```

```
X1 turbo ｼﾘｰｽﾞ ﾉ ﾊﾞｲﾄ

:0000=C3 66 00 0E 00 01 10 00 08 3E 1D D3 00 C3 9E 02 /ﾃf.......>.ﾓ.ﾃ.
:0010=C3 66 00 E3 F6 C9 C8 88 CD 1B 00 C5 06 1D ED 41 /ﾃf.●月ﾉﾈ ﾍ..ﾅ...A
:0020=C9 07 42 72 65 61 6B 00 C3 66 00 20 69 6E 20 00 /ﾉ.Break.ﾃf. in .
:0030=C3 66 00 4F 6B 00 8C 00 C3 66 00 55 6E 70 72 69 /ﾃf.Ok.▮.ﾃf.Unpri
:0040=6E 74 61 62 6C 65 20 45 72 72 6F 72 20 00 FE 30 /ntable Error .生0
:0050=D8 FE 3A 3F C9 1A 13 FE 20 C0 18 F9 1A FE 61 D8 /ﾘ生:?ﾉ..生 ﾀ.分.生aﾘ
:0060=FE 7B D0 D6 20 C9 31 00 00 2A 36 00 E9 00 40 01 /生{ﾐﾖ ﾉ1..*6...@.
:0070=40 03 40 FF FF CA 88 CB 88 FB 8B FD 8B FD 8B 00 /@.@〒〒ﾊ ﾋ 年 人 人.
:0080=EF FE EE E5 F6 FF FF 02 00 3F 20 00 01 99 10 DF /□生 月〒〒..? ...°
:0090=01 20 12 DF 01 97 77 DF CD 30 2A 01 74 04 AF CD /. .°.┐w°ﾍ0*.t.ｯﾍ
:00A0=53 02 CD 7B 35 ED 7B 7F 00 CD 82 04 AF 32 38 5C /S.ﾍ{5 {π.ﾍ▄.ｯ28¥
:00B0=01 70 17 DF 11 33 00 CD E0 04 3A 4F 5C B7 3E 2E /.p.°.3.ﾍ●.:O¥ｷ>.
:00C0=20 02 3E 20 01 91 17 DF 01 70 17 DF 21 FF FF 22 / .> .|.°.p.°!〒〒"
:00D0=85 00 11 00 FF 01 E4 1D DF 30 1D FE 04 20 10 AF /■...〒.◆.°0.生. .ｯ
:00E0=32 F9 FB CD BC 04 CD 7B 35 CD 69 05 CD 8D 05 AF /2分年ﾍｼ.ﾍ{5ﾍi.ﾍ▮.ｯ
:00F0=01 B6 7D CD 27 02 18 D4 CD FD 00 18 CF CD 55 00 /.ｶ}ﾍ'..ﾔﾍ人..ﾏﾍU.
```

図5.1 0000H 番地からのダンプ

0000H 番地から 0002H 番地までの 3 バイトに注目してください。これらは，やはりジャンプ命令のマシンコードです。ニーモニック表記では，**JP 00FAH**（X1 シリーズ），**JP 0066H**（turbo シリーズ）となります。こうして，さらに

ジャンプすると，システム初期化が行われてBASICスタートとなります。BASICのスタートのさせ方には，メモリー上のユーザー•プログラムをクリアしてしまう「**コールド・スタート**」と，クリアしない「**ホット・スタート**」の2つがありますが，X1，turbo両シリーズいずれでも**JP 0000Hは，BASICホット•スタート**となります。BASICホット•スタートへのジャンプは，ユーザーのマシン語プログラムの最後に置いて，プログラム停止に用いることができます。

5.3　絶対ジャンプ（条件付）

次に条件付ジャンプ命令JP cc, mnの説明します。ここで，ccは**条件コード(condition code)**の略で，Fレジスタのビットを参照するためのものです。条件コードには次のものがあります。

条件	読み方	意味
NZ	ノン・ゼロ	Zフラグ＝0であれば条件成立.
Z	ゼロ	Zフラグ＝1であれば条件成立.
NC	ノン・キャリー	Cyフラグ＝0であれば条件成立.
C	キャリー	Cyフラグ＝1であれば条件成立.
PO	パリティ・オッド	P/Vフラグ＝0（パリティ奇数）であれば条件成立.
PE	パリティ・イーヴン	P/Vフラグ＝1（パリティ偶数）であれば条件成立.
P	プラス	Sフラグ＝0であれば条件成立.
M	マイナス	Sフラグ＝1であれば条件成立.

図5.2　条件コードの種類

条件付ジャンプでは，条件成立のときにのみmn番地へジャンプし，条件不成立ならジャンプせずに次の命令に進みます。たとえば，

JP　NZ, mn ｛ Z＝0ならmn番地へジャンプ。
Z＝1なら次へ進む。

となります。

条件付ジャンプを利用すると，BASICの IF〜THEN〜や FOR〜NEXT に相当する処理をマシン語で実現できます。次のプログラム例では，1から100までの整数の総和を100＋99＋98＋…＋2＋1の形で計算しています。

アドレス	マシンコード	ニーモニック
D000	31　30　D0	LD SP, 0D030H
D003	21　00　00	LD HL, 0
D006	01　64　00	LD BC, 100
D009	09	ADD HL, BC
D00A	0B	DEC BC
D00B	78	LD A, B
D00C	B1	OR C
D00D	C2　09　D0	JP NZ, 0D009H
D010	E5	PUSH HL
D011	C3　00　00	JP 0000H

図5.3　1から100までの総和計算

```
MON
*M D000
:D000=31 30 D0
:D003=21 00 00
:D006=01 64 00
:D009=09
:D00A=0B
:D00B=78
:D00C=B1
:D00D=C2 09 D0
:D010=E5
:D011=C3 00 00
:D014=00
*G D000
```

```
Ok
MON
*D D000
:D000=31 30 D0 21 00 00 01 64 /10ﾐ!...d
:D008=00 09 0B 78 B1 C2 09 D0 /...×ｱｯ.ﾐ
:D010=E5 C3 00 00 00 00 00 00 /♣ﾃ......
:D018=00 00 00 00 00 00 00 00 /........
:D020=00 00 00 00 00 00 00 00 /........
:D028=00 00 00 00 00 00 BA 13 /......ｺ.
:D030=00 00 00 00 00 00 00 00 /........
:D038=00 00 00 00 00 00 00 00 /........
:D040=00 00 00 00 00 00 00 00 /........
:D048=00 00 00 00 00 00 00 00 /........
:D050=00 00 00 00 00 00 00 00 /........
:D058=00 00 00 00 00 00 00 00 /........
:D060=00 00 00 00 00 00 00 00 /........
:D068=00 00 00 00 00 00 00 00 /........
:D070=00 00 00 00 00 00 00 00 /........
:D078=00 00 00 00 00 00 00 00 /........
*█
```

図5.4　オブジェクトの実行と結果の確認

前章で学んだようにHLレジスタ・ペアを16ビットのアキュムレータとして使用しています。BCレジスタ・ペアはカウンタで100（＝0064H）から0まで **DEC BC** で1ずつ減じられます。**LD A, B** と **OR C** によりZフラグを変化させ，**JP NZ, 0D009H** では，BC＝0にならない限りD009H番地へジャンプして，加算をくり返します。BC＝0になるとループを抜けて，**PUSH HL** により（HLに得られている）結果をスタック（DO2FH番地より「上」へ）に格納してから，BASICホットスタートへジャンプして停止します。答えは，

13BAH＝5050　で正解ですね！

5.4　リロケータブルなプログラム

相対ジャンプ命令の解説をする前に，1つ大切なことについて考えておきましょう。

前節で作ったプログラムを反省します。プログラム内でスタックポインタを設定していますが，これは結果を見るための配慮なので，**LD SP, 0D030H** および **PUSH HL** を削除します。また，スタックを操作しないのでプログラム停止

を **RET** にします。このような変更をしたものが次のプログラムです。

アドレス	マシンコード	ニーモニック
D000	21 00 00	LD HL, 0
D003	01 64 00	LD BC, 100
D006	09	ADD HL, BC
D007	0B	DEC BC
D008	78	LD A, B
D009	B1	OR C
D00A	C2 06 D0	JP NZ, 0D006H
D00D	C9	RET

図5.5 絶対ジャンプを含むプログラム例

これでは結果は見れませんが，ともかくプログラム実行直後のHLレジスタ・ペアには答えが得られているはずですね。

ここで次のことを考えてみます。今，D000H番地からには，すでに大切なプログラムが入っていて，私たちのプログラムを別の場所(例えば，E000H番地から）に格納しなければならないとしたらどうでしょうか？ **マシンコードに何の変更も加えず**，そのままE000H番地から書き込んでみます(図5.6)。

さて，このプログラムを **＊ G E000** により実行した場合に期待通りに動くのでしょうか？ E000AH番地に注目してください。**JP NZ, 0D006H** という命令がありますね。CPUはこれを実行すると，条件成立ならD006H番地へジャンプしてしまいますが，今度はこのあたりには別のプログラムが入っているかもしれず，何が起きるかわかりませんね。ということは，**JP NZ, 0D006H** で飛び先を固定しているために，このプログラムはD000H番地から格納したときにのみ正しい働きをし，格納番地の変更はできないということです。

これに対して，もしD00AH番地のジャンプ命令が「4番地だけ前へジャンプ

せよ」の形で書かれていたとしたらどうでしょう。これなら，たとえば格納場所がE000H番地から移動しても，E00AH番地の4番地前であるE006H番地へジャンプして，全く同じ働きができるようになりますね。このように，別の場所に配置しても同様に動くプログラムを**リロケータブル(relocatable＝再配置可能)**である，あるいは**ポジション・インデディペンデント(position independent＝位置独立)**であるといいます。(条件付の)絶対ジャンプ命令を含んだ「総和プログラム」は，リロケータブルではないプログラムの例です。

アドレス	マシンコード	ニーモニック
E000	21 00 00	LD HL, 0
E003	01 64 00	LD BC, 100
E006	09	ADD HL, BC
E007	0B	DEC BC
E008	78	LD A, B
E009	B1	OR C
E00A	C2 06 D0	JP NZ, 0D006H
E00D	C9	RET

図5.6 格納場所の変更

5.5 相対ジャンプ命令

これから説明する相対ジャンプ命令は，リロケータブルなプログラムを書こうとするときは不可欠なものです。

相対ジャンプ命令は**JR**というOPコードをもち，無条件と条件付の2系列があります。

無条件……… JR e

条件付…… JR C, e / JR NC, e / JR Z, e / JR NZ, e

条件付の方は，絶対ジャンプと異なって条件コードは4種類しか許されていません。

これらに共通しているのは，オペランドに書かれるeという1バイト値で**相対アドレス**とよばれます。相対アドレスは，JR命令が書かれている番地からジャンプ先までの隔たり（ディスプレイスメント）を示す「1バイト符号つき整数」です。たとえば，JR eという命令を実行すると，PC←PC+e（PCはプログラム・カウンタ）が行われ，相対ジャンプ命令JR eが書かれている番地（PCの現在値）からe番地離れた番地に制御が移ります。ただし，注意しなければならないのは，約束事として，相対ジャンプ命令をオブジェクトに翻訳するとき，eの値はe−2として変換されるという点です。たとえば，

JR −4 (e) ═アセンブル⇒ 18H，FAH (e−2＝−6)

となります。そして，eの値はオブジェクトに落としたe−2の方が「1バイト符号つき整数」の範囲である−128〜＋127に入るように指定しなければなりません。こうして，**eの許される範囲**は2だけずらした**−126〜＋129**の間になることに注意しましょう。

［注］ 2だけずれるのは余り気持ちよくありませんが，要するに相対アドレスを考える基準点がどこか？ ということの違いです。ニーモニックの方では，JR命令の書かれているアドレスを基準とするのに対し，オブジェクトの方では相対ジャンプの2バイト命令の次のアドレスを基準とするからです。

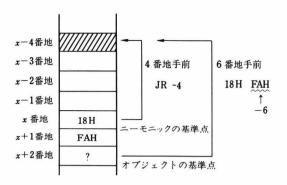



図5.7　相対アドレスの基準点

ただし，こうした「約束事」はアセンブラにより異なることもあり，オブジェクトの相対アドレスとニーモニックの相対アドレスを一致させる流儀もあります。実際，文献［2］では **JR e** のオブジェクトを18H，eとしていますが，ここではシャープの資料［1］に従って，本文中で述べたような「2だけずらす」約束を採用します。

5.6　相対ジャンプでリロケータブルに！

では，条件付の相対ジャンプ **JR　NZ, e** を用いて，「1から100までの総和プログラム」をリロケータブルにしてみましょう(図5.8)。

JR　NZ, −4 では相対アドレス e=−4 ですから，オブジェクトに直すとき e−2=−6 を行い 06H の補数 FAH を得ます。

実は，このプログラムはリロケータブルなサブルーチンです。後節で説明することを先取り（サブルーチンの呼び出し）して，今までとは変わった方法で実行してみましょう。次のプログラムを続いて書き込んだ後，**＊G　D100H** により実行してください。答は，D110H 番地と D111H 番地の2バイトに上下位逆転して得られます(図5.9，図5.10)。

アドレス	マシンコード	ニーモニック
D000	21 00 00	LD HL, 0
D003	01 64 00	LD BC, 100
D006	09	ADD HL, BC
D007	0B	DEC BC
D008	78	LD A, B
D009	B1	OR C
D00A	20 FA	JR NZ, −4
D00C	C9	RET

図5.8 相対ジャンプでリロケータブルに！

アドレス	マシンコード	ニーモニック
D100	CD 00 D0	CALL 0D000H
D103	22 10 D1	LD (0D110H), HL
D106	C9	RET

図5.9 結果確認のためのプログラム

```
MON
*M D000
:D000=21 00 00
:D003=01 64 00
:D006=09
:D007=0B
:D008=78
:D009=B1
:D00A=20 FA
:D00C=C9
:D00D=00
*M D100
:D100=CD 00 D0
:D103=22 10 D1
:D106=C9
:D107=00
*G D100
*D D110 D111
:D110=BA 13 00 00 00 00 00 00 /コ.......
*■
```

図5.10 実行と確認

5.7　特別なジャンプ命令

Z80のジャンプ命令には，以上のほかに特別なジャンプが4種類あります。

```
JP  (HL)
JP  (IX)
JP  (IY)
DJNZ e
```

の4種です。

JP (HL)を初めとする最初の3つは，各16ビット・レジスタの内容をプログラム・カウンタPCにロードすることによるジャンプ命令です。

たとえば，JP (HL)ではPC←HLが行われるので，PC=1234H，HL=5678HのときJP (HL)を実行すると，PC=5678Hとなり，CPUは次に5678H番地の命令を実行します。

DJNZ eは，特別な相対ジャンプ命令で，eは－126～＋129の範囲の相対アドレスです。何が特別かというと，この命令はBレジスタをカウンタとして，Bレジスタの内容を1だけ減じ，B≠0でなければPC←PC+eを行いジャンプし，B=0であれば次の命令を実行するというものです。この命令は便利なので，Z80のマシン語プログラムではよく用いられます。

次のプログラムは，DJNZ命令で変化していくBレジスタの様子を，D100H番地からD1FFH番地まで順に格納するものです。

アドレス	マシンコード	ニーモニック
D000	21 00 D1	LD HL, 0D100H
D003	06 00	LD B, 0
D005	70	LD (HL), B
D006	23	INC HL
D007	10 FC	DJNZ −2
D009	C9	RET

図5.11 DJNZ eでのBレジスタを追う！

```
MON
*M D000
:D000=21 00 D1
:D003=06 00
:D005=70
:D006=23
:D007=10 FC
:D009=C9
:D00A=00
*G D000

*D D100
:D100=00 FF FE FD FC FB FA F9 /・〒生人円年秒分
:D108=F8 F7 F6 F5 F4 F3 F2 F1 /時日月火水木金土
:D110=F0 EF EE ED EC EB EA E9 /▒□▙▃▘▖▚▝
:D118=E8 E7 E6 E5 E4 E3 E2 E1 /╳◣◢♣◆♥♠○
:D120=E0 DF DE DD DC DB DA D9 /●▪゛ンワロレル
:D128=D8 D7 D6 D5 D4 D3 D2 D1 /リラヨユヤモメム
:D130=D0 CF CE CD CC CB CA C9 /ミマホヘフヒハノ
:D138=C8 C7 C6 C5 C4 C3 C2 C1 /ネヌニナトテツチ
:D140=C0 BF BE BD BC BB BA B9 /タソセスシサコケ
:D148=B8 B7 B6 B5 B4 B3 B2 B1 /クキカオエウイア
:D150=B0 AF AE AD AC AB AA A9 /ーッョュャォェゥ
:D158=A8 A7 A6 A5 A4 A3 A2 A1 /ィァヲ・、」「。
:D160=A0 9F 9E 9D 9C 9B 9A 99 / ╲╭╯╮╰┌└
:D168=98 97 96 95 94 93 92 91 /┘┐┼┤┬┴├│
:D170=90 8F 8E 8D 8C 8B 8A 89 /─╱█▉▊▋▌▍
:D178=88 87 86 85 84 83 82 81 /▏▇▆▅▄▃▂▁

*D
:D180=80 7F 7E 7D 7C 7B 7A 79 /_π‾}|{zy
:D188=78 77 76 75 74 73 72 71 /xwvutsrq
:D190=70 6F 6E 6D 6C 6B 6A 69 /ponmlkji
:D198=68 67 66 65 64 63 62 61 /hgfedcba
:D1A0=60 5F 5E 5D 5C 5B 5A 59 /`_^]¥[ZY
:D1A8=58 57 56 55 54 53 52 51 /XWVUTSRQ
:D1B0=50 4F 4E 4D 4C 4B 4A 49 /PONMLKJI
:D1B8=48 47 46 45 44 43 42 41 /HGFEDCBA
:D1C0=40 3F 3E 3D 3C 3B 3A 39 /@?>=<;:9
:D1C8=38 37 36 35 34 33 32 31 /87654321
:D1D0=30 2F 2E 2D 2C 2B 2A 29 /0/.-,+*)
:D1D8=28 27 26 25 24 23 22 21 /('&%$#"!
:D1E0=20 1F 1E 1D 1C 1B 1A 19 / .......
:D1E8=18 17 16 15 14 13 12 11 /........
:D1F0=10 0F 0E 0D 0C 0B 0A 09 /........
:D1F8=08 07 06 05 04 03 02 01 /........
*■
```

図5.12 実行と確認

注意していただきたいのは，Bレジスタの初期値を00Hにしておくと，**DJNZ e**でB←B-1となったとき，FFHが次の値であることです。すなわち，1バイトの範囲では00Hは256と同等なわけで，**DJNZ e**では1回～256回までの繰り返しが実現できます。

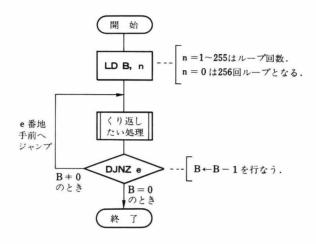



図5.13　DJNZ eでのループ

5.8　ラベルについて

ジャンプ命令の飛び先を示すのに**ラベル**を使うとわかりやすくなります。ラベルもアセンブリ言語の構成要素の1つで，ふつう英字で始まる英数字6字以内が用いられます。ラベルは，命令のOPコードの前に書かれ，ラベルの後には：（コロン）を必ず書きます。

ラベル：**OPコード**　**オペランド**

という形式になります。このように，ラベルを定義しておけば，そのプログラム内の命令のオペランドにアドレス値のかわりにラベルを使うことができます。

たとえば，図5.11のソース・プログラムは次のようにLOOP（ループ）というラベルを使用すると見やすくなります。

ソース・プログラム	
	LD HL, 0D100H
	LD B, 0
LOOP：	LD (HL), B
	INC HL
	DJNZ LOOP
	RET

図5.14 ラベルを使って見やすくする

5.9 BASICでのサブルーチン

BASIC言語でプログラムを作るとき，構造的にまとまりをもった処理(モジュール)や，何度も繰り返し使われる手続きなどを**サブルーチン**にすることは，よく行われます。

次に挙げたプログラム例では，GCMSUBというラベルをつけた210行以降がサブルーチンで，M1とN1の2数の最大公約数を変数GCMに入れて返します。メインルーチン（100行〜190行）は，3個の数A，B，Cの最大公約数を求めるためにGCMSUBを2度呼び出して目的を果たしています。

```
100 REM  3コ ノ カズ ノ サイダイ コウヤクスウ
110 '
120 INPUT "a = ",A
130 INPUT "b = ",B
140 INPUT "c = ",C
150 M1=A   : N1=B : GOSUB "GCMSUB"
160 M1=GCM : N1=C : GOSUB "GCMSUB"
170 PRINT "GCM(a,b,c) =";GCM
180 PRINT
190 GOTO 120
200 '
210 LABEL "GCMSUB"
220 '
230   IF N1>M1 THEN SWAP M1,N1
240   R1=(M1 MOD N1)
250   WHILE R1<>0
260     M1=N1 : N1=R1 : R1=(M1 MOD N1)
270   WEND
280   GCM=N1
290   RETURN
300 '
```

```
RUN
a = 20
b = 12
c = 36
GCM(a,b,c) = 4

a = 9
b = 6
c = 4
GCM(a,b,c) = 1

a = █
```

図5.15 BASICのサブルーチンの例

大切なことは，サブルーチンが**GOSUB**文で呼び出されるごとに，どこに戻ればよいかが「システムのどこかに」記憶され，**RETURN**文を実行すると**GOSUB**の次の命令に戻れることです。したがって，上のプログラムで，1回目の**GCMSUB**の呼び出しと，2回目のそれとで**RETURN**後に戻る場所が異なります。これが単なる**GOTO**文によるジャンプと異なる点ですね。

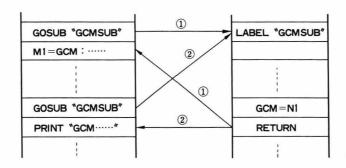



図5.16 サブルーチン実行のしくみ

5.10 マシン語でのサブルーチン

マシン語プログラムにおけるサブルーチン実行のしくみもBASICの場合と全く同様の原理で行われます。

マシン語サブルーチンを呼び出すためのマシン語命令の基本形は，

ニーモニック	マシンコード
CALL mn	CD n m

です。mnは，サブルーチンの先頭番地を表す2バイト数値で，mは上位バイト，nは下位バイトです。

CPUはCALL命令を実行すると，その次の命令のある番地(CALL mn実行後のプログラム・カウンタPCの値)をスタックに自動的にプッシュした後(それに伴って，スタック・ポインタSPは自動的に－2される！)，mn番地にジャンプします。

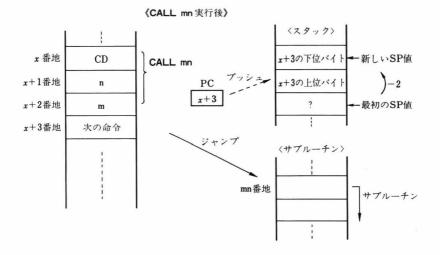



図5.17　マシン語サブルーチンの実行のしくみ（I）

上の図で，戻り番地である$x+3$の値（番地ですから2バイト値）がスタックに保存される点が大切ですよ！

さて，マシン語サブルーチンの出口に

ニーモニック	**マシンコード**
RET	**C9**

で表される**RET**命令を書いておきますと，CPUがこれを実行したとき，まずそのときのスタックより2バイトをプログラム・カウンタ（PC）にポップするという動作をします（同時に**SP**の値は＋2される）。サブルーチン内で**SP**の値を変更してない限り，スタックの一番「上」には戻り番地が積まれているはずですから，これをPCにポップすれば**CALL mn**の次の番地にジャンプできるわけですね。

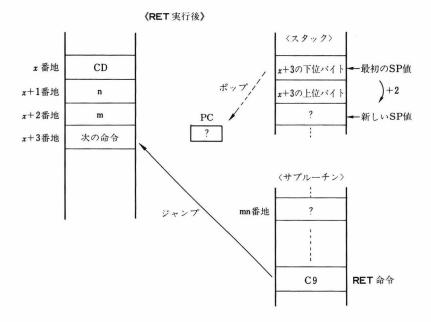



図5.18 マシン語サブルーチンの実行のしくみ（II）

マシン語サブルーチンの実行のしくみ（呼び出しとリターン）について，ご理解いただけましたか？　以上の過程を，Z80 にはない命令 **PUSH PC** や **POP PC** を混じえて表わすと次のようになります。

```
CALL  mn   = { PUSH  PC
             { JP  mn
RET        =  POP  PC
```

5.11 オマジナイ C9H の秘密

マシン語サブルーチンの実行原理がわかったところで，長い間の懸案事項であった「なぜ C9H がプログラム停止のオマジナイであるか？」に解答を与えておきましょう。

実は，モニターのGコマンドを実行すると，モニター処理（＊を表示しコマンド入力を待つ）が開始される番地がスタックにプッシュされた後，Gコマンドで指定された先頭番地へジャンプするようになっています。さて，私たちユーザーのプログラム中でSPの値を最終的に変更していなければ，プログラムの出口にRET（＝C9H）を置いておくと，RETを実行したとき，スタックに積まれていたモニターへの戻り番地がPCにポップされて，めでたくモニターへ戻れるわけですね。こうして，プログラム実行が停止されます。

ところで，私たちはもう1つ JP 0000H による停止法を使ってきました。これを採用したプログラムを注意深く見ていただくとわかりますが，プログラム中で LD SP, mnやPUSH 命令を実行していたはずです。こうすると，SPの値はずれてしまいRETを行っても，モニターへの戻り番地がポップされませんね。ですから，BASICのホット・スタートへジャンプして，プログラムを停止させたのです。これで疑問は氷解したことでしょう！

5.12 CALL 命令

では，Z80 マシン語におけるサブルーチン呼び出しの命令についてまとめます。呼び出しは英語で call（コール）というので，これをニーモニックのOPコードとして採用したCALL命令とよばれる一群の命令です。

ジャンプ命令と同様に，CALL命令には無条件にサブルーチンの呼び出しをする命令と，フラグを参照する条件付CALL命令とがあります。

無条件 CALL 命令…CALL mn
条件付 CALL 命令…CALL cc, mn
（cc は条件コード）

いずれも mn は，サブルーチンの先頭番地を示す2バイト値で，m は上位バイト，n は下位バイトです。ジャンプ命令と異なり相対番地を用いた CALL 命令は Z80 にはありません。

条件付 CALL 命令での cc は条件コードで，JP 命令と同様に以下のものが

許されます。

- NZ (ノン・ゼロ)
- Z (ゼロ)
- NC (ノン・キャリー)
- C (キャリー)
- PO (パリティ・オッド)
- PE (パリティ・イーヴン)
- P (プラス)
- M (マイナス)

次の例は，X1 の HuBASIC 内にある IOCS (入出力制御システム)中，カセットテープからの読み込み照合 (ベリファイ) をする処理ルーチンです。

アドレス	マシンコード	ニーモニック
0BAE	D5	PUSH DE
0BAF	16 08	LD D, 08H
0BB1	F3	DI (注1)
0BB2	CD C7 0D	CALL 0DC7H
0BB5	D4 20 0D	CALL NC, 0D20H
0BB8	D4 3A 0C	CALL NC, 0C3AH
0BBB	18 D1	JR −45 (注2)

(注1) 割り込み禁止命令 (第10章10.9節参照).
(注2) ジャンプ先は0B8EH番地.

図5.19 データ・ブロック VERIFY ルーチン

(X1 HuBASIC より)

X1シリーズのユーザの方は，モニターよりダンプして見てください。turbo BASICでは異なりますが，turboの方もディスクBASIC CZ-8FB01を起動すれば見られます。

0DC7H番地からサブルーチン（カセットのモーターON）を実行して返ってくると，テープの状態によってCyフラグが変化しています。そこで，0BB5H番地には「Cy＝0ならば0D20H番地のサブルーチン（テープのヘッダー部分の読み取り）をコールせよ。」という条件付**CALL**命令が書かれています。

5.13 RET命令

さて，サブルーチンの呼び出しには，サブルーチンからの**リターン（復帰）**が対となっているのが普通で，returnの省略形である**RET**をOPコードとする**RET**命令が使われます。**RET**命令にも，無条件**RET**と条件付**RET**があります。

無条件**RET**命令……**RET**
条件付**RET**命令……**RET cc**

ccは条件コードで，**CALL**命令の条件コードと同じものが許されます。

次は，やはりX1 HuBASICのIOCS内の例で，テープ走行とキー入力状態のチェックをするサブルーチンです(図5.20)。

0330H～0333H番地では，第6章で説明する入力命令を用いて，カセット・レコーダ関係のデータをAレジスタに受けとっています。次の**AND 01H**がマスクをかける論理演算で，Aレジスタの第0ビット（最も右のビット）が0か1かをZフラグで判別しています。実は，このビットが1ならテープ走行中，0なら停止中を意味しますので，**RET NZ**はこの場合「テープ走行中ならリターン」と解釈できます。Zフラグが立っていると（つまり，テープが止まっていると），**RET NZ**は素通りして，0337H番地からの処理（キー入力関係のデータを受けとる）を実行した後，0345H番地の**RET**でサブルーチンより戻るわけです。

アドレス	マシンコード	ニーモニック
0330	3E 1A	LD A, 1AH
0332	DB 01	IN A,(01H)（注1）
0334	E6 01	AND 01H
0336	C0	RET NZ
0337	3E E6	LD A, 0E6H
0339	CD FE 0D	CALL 0DFEH
033C	CD 49 0B	CALL 0B49H
033F	CD 49 0B	CALL 0B49H
0342	FB	EI（注2）
0343	FE 03	CP 03H
0345	C9	RET

（注1）　入力命令の一種（第6章6.2節参照）.
（注2）　割り込み許可命令（第10章10.9節参照）.

図5.20　テープの走行チェック

（X1 HuBASIC より）

5.14　マシン語サブルーチンの作成

では，簡単なマシン語サブルーチンを作ってみましょう。図5.15で述べたBASICサブルーチンのマシン語版を作ってみます。

仕様は以下の通りです。

仕様：① 3個の数は1～255（10進）の範囲で，対応する16進数をモニターよりD040H，D041H，D042H の各番地にセットしておく．
② 最大公約数はD044H番地に格納される．
③ メインルーチンはD000H番地から，サブルーチンはD020H番地から配置する．
④ サブルーチンは，Dレジスタ，Eレジスタにセットされた2数の最大公約数を求め，結果をAレジスタに入れて戻る．

図5.21　3個の数の最大公約数を求める

マシン語サブルーチン

この仕様に基づいて，次のようなソースプログラムを作り，ハンド・アセンブルしました。

メイン・ルーチンのプログラム

アドレス	マシンコード	ニーモニック
D000	21　40　D0	LD HL, 0D040H
D003	56	LD D, (HL)
D004	23	INC HL
D005	5E	LD E, (HL)
D006	CD　20　D0	CALL 0D020H
D009	57	LD D, A
D00A	23	INC HL
D00B	5E	LD E, (HL)
D00C	CD　20　D0	CALL 0D020H
D00F	23	INC HL
D010	23	INC HL
D011	77	LD (HL), A
D012	C9	RET

サブルーチンのプログラム

アドレス	マシンコード	ニーモニック
D020	7A	LD A, D
D021	BB	CP E
D022	30 02	JR NC, MOD1
D024	53	SWAP : LD D, E
D025	5F	LD E, A
D026	7A	MOD1 : LD A, D
D027	93	LP1 : SUB E
D028	30 FD	JR NC, LP1
D02A	83	ADD A, E
D02B	28 09	JUDGE : JR Z, EXIT
D02D	53	LD D, E
D02E	5F	LD E, A
D02F	7A	MOD2 : LD A, D
D030	93	LP2 : SUB E
D031	30 FD	JR NC, LP2
D033	83	ADD A, E
D034	18 F5	JR JUDGE
D036	7B	EXIT : LD A, E
D037	C9	RET

図5.22 メイン・ルーチンのプログラム

サブルーチンのプログラム

プログラムの考え方の説明は後まわしにして，Mコマンドで入力してください。正しく入力されれば，ダンプすると次のようになるはずです。

```
MON
*D D000 D037
:D000=21 40 D0 56 23 5E CD 20 /!@ミV#^ヽ
:D008=D0 57 23 5E CD 20 D0 23 /ミW#^ヽ ミ#
:D010=23 77 C9 00 00 00 00 00 /#wJ.....
:D018=00 00 00 00 00 00 00 00 /........
:D020=7A BB 30 02 53 5F 7A 93 /zサ0.S_z┬
:D028=30 FD 83 28 09 53 5F 7A /0人▄(.S_z
:D030=93 30 FD 83 18 F5 7B C9 /┬0人▄.火{J
*■
```

図5.23　プログラム入力後のダンプ

実行するには，D040H 番地から3バイト分のデータをセットしなければなりません。次の例は，1回目のものが20(＝14H)，12(＝0CH)，36(＝24H)の最大公約数を求めていて，D044H 番地に答4(＝04H)が得られています。2回目のものは，9と6と4の最大公約数1を求めています。

```
*M D040
:D040=14
:D041=0C
:D042=24
:D043=00
*G D000
*M D044
:D044=04
*M D040
:D040=09
:D041=06
:D042=04
:D043=00
*G D000
*M D044
:D044=01
*■
```

図5.24　実行例

皆さんも，いろいろな値を入力して試してください。

プログラムのうち，メインルーチンはわかりやすいと思います。HL レジスタ・ペアをデータのポインタとして使用し，INC HL によりポインタを進めています。

サブルーチン（D020H番地〜）の方が少しとっつきにくいでしょう。相対ジャンプを用いたので，ラベルを使用しておきました。まず大切なのは，考え方の上では，図5.15で述べたBASICのサブルーチンと全く同様だということです。BASICの方で**GCMSUB**と名づけたサブルーチンをご覧ください。変数M1の役割をDレジスタ，変数N1の役割をEレジスタが果たしていると見て，マシン語サブルーチンを読むと意味がわかります。

MOD1，**MOD2**とラベルをつけた番地からの処理は整数除算での余りを求めるものです（BASICの**MOD**演算にあたる)。減算のくり返し（LP1，LP2がループ先の番地）により実現しています。Cyフラグが立つまで減算するので，ループから出たときは1回分引きすぎとなっています。そこで**ADD A，E**により補正して，Aレジスタに余りを得ます。BASICの**WHILE〜WEND**のループは，マシン語では**JUDGE**というラベル以降の処理で実現しています。

サブルーチンの出口は**EXIT**以降で，Eレジスタに得られている最大公約数をAレジスタにセットしてメインルーチンへ戻ります。

相対番地をマシンコードに変換する練習を2つしておきましょう。まず，

JR　NC，MOD1

ですが，**MOD1**の示すD026H番地から命令の書かれているD022H番地を引いて，e＝D026H－D022H＝04Hがニーモニックでの相対番地。マシンコードに直すにはe－2＝04H－2＝02Hとなります。次に

JR　JUDGE

の場合です。**JUDGE**とラベルがつけてあるのはD02BH番地。命令の書かれているのはD034H番地。

引き算すると，

e＝D02BH－D034H＝－9

e－2＝－9－2＝－11＝F5H（補数）となるわけですね。よろしいですか？

5.15　リスタート命令

実は，サブルーチンの呼び出しには，**CALL** 命令以外に**リスタート命令**とよばれるものもあります。**CALL** 命令では，サブルーチンのアドレスを各々指定しなければなりませんでしたが，リスタート命令ではきまったアドレスが呼び出される点が違います。いわば「既製品のサブルーチン呼び出し」といえます。

リスタート命令は，**RST**（ReSTart の略）という OP コードと1個のオペランドを持ちます。

オペランドによって次の8種類があります。

ニーモニック	マシンコード	機　　能
RST 00H	C7	CALL 0000H と同じ．
RST 08H	CF	CALL 0008H と同じ．
RST 10H	D7	CALL 0010H と同じ．
RST 18H	DF	CALL 0018H と同じ．
RST 20H	E7	CALL 0020H と同じ．
RST 28H	EF	CALL 0028H と同じ．
RST 30H	F7	CALL 0030H と同じ．
RST 38H	FF	CALL 0038H と同じ．

図5.25　リスタート命令

これらの命令で呼び出される番地のあたりをダンプしてみます。X1の HuBASIC および turbo BASIC では次のようになっています。

```
MON
*D 0000
:0000=C3 FA 00 C3 7C 01 50 28 /テ秒.テ|.P(
:0008=C3 D3 03 C3 83 04 18 03 /テモ.テ▄.%.
:0010=C3 D3 03 C3 BC 04 00 18 /テモ.テシ...
:0018=C3 D3 03 C3 9D 02 00 27 /テモ.テ゛..'
:0020=C3 D3 03 C3 07 0E 07 20 /テモ.テ...
:0028=C3 D3 03 C3 AA 0A 0D 37 /テモ.テェ..7
:0030=C3 D3 03 C3 CA 0A 00 FF /テモ.テハ..テ
:0038=C3 D3 03 C3 75 0B C3 79 /テモ.テu.テy
:0040=0B C3 9A 0B C3 9E 0B C3 /.テ┌.テ╭.テ
:0048=AE 0B C3 30 03 C3 88 09 /ョ.テ0.テ|.
:0050=00 00 46 03 D3 03 D3 03 /..F.モ.モ.
:0058=D3 03 D3 03 D3 03 D3 03 /モ.モ.モ.モ.
:0060=D3 03 D3 03 D3 03 C3 FA /モ.モ.モ.テ秒
:0068=00 63 0E 63 0E 8B 07 1E /.c.c.▌..
:0070=07 63 0E F1 07 A8 07 F7 /.c.土.ィ.日
:0078=07 14 03 A1 03 1B 07 BF /....▪...ソ
*■
```

図5.26 RST 命令でのジャンプ先（X1）

```
KMODE 0
Ok
MON
*D 0000
:0000=C3 66 00 0E 00 01 10 00 /テf......
:0008=08 3E 1D D3 00 C3 9E 02 /.>.モ.テ╭.
:0010=C3 66 00 E3 F6 C9 C8 88 /テf.♥月ノネ|
:0018=CD 1B 00 C5 06 1D ED 41 /ヘ..ナ..▞A
:0020=C9 07 42 72 65 61 6B 00 /ノ.Break.
:0028=C3 66 00 20 69 6E 20 00 /テf. in .
:0030=C3 66 00 4F 6B 00 8C 00 /テf.Ok.▌.
:0038=C3 66 00 55 6E 70 72 69 /テf.Unpri
:0040=6E 74 61 62 6C 65 20 45 /ntable E
:0048=72 72 6F 72 20 00 FE 30 /rror .生0
:0050=D8 FE 3A 3F C9 1A 13 FE /リ生:?ノ..生
:0058=20 C0 18 F9 1A FE 61 D8 / タ.分.生aリ
:0060=FE 7B D0 D6 20 C9 31 00 /生{ミヨ ノ1.
:0068=00 2A 36 00 E9 00 40 01 /.*6.▘.@.
:0070=40 03 40 FF FF CA 88 CB /@.@テテハ|ヒ
:0078=88 FB 8B FD 8B FD 8B 00 /|年▌人▌人▌.
*■
```

図5.27 RST 命令でのジャンプ先（turbo）

多くは，**JP** 命令が並んでいますね。X1 HuBASIC では，**JP 03D3H** がほとんどですが，03D3H からは「割り込み処理からのリターン」という特別な処理が書かれていて，後述する（第10章10.6節参照）割り込みを想定して作られ

ているようです。また，turbo BASIC では，**JP 0066H** が多いですが，これはシステム初期化ルーチンへのジャンプです。いずれも，**RST** 命令は BASIC で積極的に利用してないようですね。

ですから，私たちユーザーが使える余地は十分にありそうですね。**CALL** 命令は3バイトかかりますが，**RST** 命令はわずか1バイトで済みますから便利です。たとえば，**RST 38H**（マシンコード FFH）を利用してみましょう。

5.16　ブレーク・ポイントへの利用

あらかじめ0038H 番地から3バイトに実行番地をジャンプ命令で書いておきます。たとえば，D100H 番地にとばしたいなら次のようにします。

```
MON
*M 0038
:0038=C3
:0039=00
:003A=D1
:003B=C3
*
```

図5.28　**RST** 38H でのジャンプ先の設定

そして，D100H 番地のプログラムを実行させたい所に1バイト FFH を書き込みますと，その箇所が実行されたときに，**RST 38H** でまず 0038H 番地へジャンプし，そこにある **JP 0D100H** を実行して再び D100H 番地へジャンプして目標とするプログラムが実行されるわけですね。

よく，BASIC プログラムの要所に **STOP** 文を書いてそこで実行を一時停止し，変数内容を調べたりしますが，**RST** 命令を上手に使うと，マシン語プログラムでのデバッグやプログラムの追跡に利用できます。

たとえば，D100H 番地からは，スタックを設定して，AF，BC，DE，HL の順に **PUSH** して内容を見るためのプログラムを書いておきましょう。

```
*M D100
:D100=31 30 D1
:D103=F5
:D104=C5
:D105=D5
:D106=E5
:D107=C3 00 00
:D10A=00
*■
```

図5.29 D100H 番地からの処理

題材として，5.14節で見た「3個の数の最大公約数を求めるプログラム」を使います。1回目の **CALL 0D020H** を実行した直後，D009H 番地でのレジスタ内容を見るため，ここに FFH を書き込みます（これを**ブレーク・ポイントの設定**といいます）。さて，データ 14H (＝20)，0CH (＝12)，24H (＝36) をセットして *** G D000**で実行すると……。

```
*D D000 D037
:D000=21 40 D0 56 23 5E CD 20 /!@ミV#^ヘ
:D008=D0 57 23 5E CD 20 D0 23 /ミW#^ヘ ミ#
:D010=23 77 C9 00 00 00 00 00 /#wノ.....
:D018=00 00 00 00 00 00 00 00 /........
:D020=7A BB 30 02 53 5F 7A 93 /zサ0.S_zT
:D028=30 FD 83 28 09 53 5F 7A /0人▄(.S_z
:D030=93 30 FD 83 18 F5 7B C9 /T0人▄.火{ノ
*M D009
:D009=FF
:D00A=23
*M D040
:D040=14
:D041=0C
:D042=24
:D043=00
*G D000
```

図5.30 最大公約数プログラムの追跡

RST 38H を経て，D100H 番地からのプログラムが実行され，BASIC ホットスタートへジャンプして停止しました。では，D100H 番地からダンプしましょう。

```
Ok
MON
*D D100
:D100=31 30 D1 F5 C5 D5 E5 C3 /104火ナユ♣テ
:D108=00 00 00 00 00 00 00 00 /........
:D110=00 00 00 00 00 00 00 00 /........
:D118=00 00 00 00 00 00 00 00 /........
:D120=00 00 00 00 00 00 00 00 /........
:D128=41 D0 04 08 03 10 51 04 /Aミ....Q.
:D130=00 00 00 00 00 00 00 00 /........
:D138=00 00 00 00 00 00 00 00 /........
:D140=00 00 00 00 00 00 00 00 /........
:D148=00 00 00 00 00 00 00 00 /........
:D150=00 00 00 00 00 00 00 00 /........
:D158=00 00 00 00 00 00 00 00 /........
:D160=00 00 00 00 00 00 00 00 /........
:D168=00 00 00 00 00 00 00 00 /........
:D170=00 00 00 00 00 00 00 00 /........
:D178=00 00 00 00 00 00 00 00 /........
*■
```

図5.31 レジスタ内容を調べる

スタックにレジスタ内容が積まれていますね。

この結果より次がわかります。

Aレジスタ ＝ 04H …20と12の最大公約数

Fレジスタ ＝ 51H

Bレジスタ ＝ 10H

Cレジスタ ＝ 03H

Dレジスタ ＝ 08H

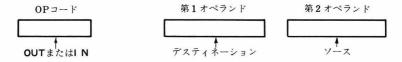
Eレジスタ ＝ 04H

Hレジスタ ＝ D0H ⎫
Lレジスタ ＝ 41H ⎭ HL＝D041H

このようにして,少しずつマシン語プログラムの動作追跡ができるわけです。

第6章　入出力と画面制御

本章で扱う命令は，I/Oポートをアクセスするためのものです。I/Oポートについては，2.3節を参照してください。CPUからI/Oポートにデータを出力する命令は**出力命令**といい，ニーモニックで**OUT**というOPコードを持ちます。逆にI/OポートからCPUへデータを入力する命令は**入力命令**とよばれ，**IN**というOPコードで表わされます。これを総称して**入出力命令**とよんでいます。

ニーモニックの形式は，**LD**命令と似ていて2つのオペランドを持ち，第1オペランドにはデスティネーション，第2オペランドにはソースが書かれます。

I/Oポートには，X1シリーズのさまざまな入出力機器が接続されているので，これらをマシン語で制御するためには，入出力命令は欠かせません。ただし，I/Oアドレスの何番地に何が接続されているかの知識が必要です。X1シリーズ(turboを除く)に関する，そのようなハードウェアの知識に関しては拙著

『X1マシン語活用百科』(産業報知センター刊)

で解説しておきましたので，興味のある方はそちらを参照してください。

本章ではI/Oポート・アクセスの例として，文字を表示する「テキスト画面」制御の方法を説明します。具体例を通じ，X1独特の入出力命令の使い方をマスターしましょう。

6.1 OUT 命令

まず出力命令から見ましょう。次の形のものがあります。

{ **OUT (C), r** … r は A, B, C, D, E, H, L の各レジスタ。
　OUT (n), A … n は 1 バイト値。

r や **A** で示されるのは，データを送り出す側（ソース）となる CPU のレジスタです。(C)や(n)で示されるのが，データの送り先(デスティネーション)の I/O ポートを表わします。

ここで，**LD (HL), A** の形のロード命令を思い出してください。(HL)は，HL レジスタの内容が示すメモリーの番地を意味していましたね。その連想で考えると，たとえば **OUT (C), A** の(C)は「C レジスタの内容が示す I/O アドレス」となりそうですが……。多くの Z80 の解説書ではそのように説明されていて，それは「ある意味で」正しい説明です。

確かに，**OUT (C), A** を実行すると，C レジスタの内容がアドレス・バスの下位 8 ビット(A_7～A_0)に出力されて I/O ポートがアクセスされます。しかし，もう 1 つ大切なことが，このときに起こるのです。それは，**この命令を実行する直前の B レジスタの内容**がアドレス・バスの上位 8 ビット (A_{15}～A_8) に出力され，Z80 では「潜在的に」I/O ポートを16ビットでアドレス指定しているのです (参考文献［1］参照)。この事情はメモリーのアドレス指定と全く同様です。ただし，Z80 のふつうの使い方では I/O アドレスは 0 ～255までを用い，上位 8 ビットの情報は無視するようにします。ニーモニックに B レジスタが登場しないのも，それがふつうだからでしょう。

しかし X1 および turbo シリーズにおいては事情が異なります。これらの機種では I/O ポートをアクセスするときの上位 8 ビットのアドレス指定が有効になるよう設計されていて，I/O アドレスは 0 番地から65535番地まで(16進では 0000H～FFFFH)使われます。したがって，ニーモニック上では隠れている B レジスタの指定が大切になるわけですね。

たとえば，X1 シリーズにおいて I/O アドレス 48CFH 番地に，データ 41H

を出力するのに**OUT (C),A**を使うなら，次のようにします。

```
LD  BC, 48CFH
LD  A, 41H
OUT (C), A
```

アドレス指定が**LD C,0CFH**では不十分であることをご理解ください。

以上はBCレジスタ・ペアによるI/Oアドレス指定の話でしたが，もう1つの形の出力命令**OUT (n),A**では事情が違います。nは，00H～FFHの1バイト値を指定し，ふつうのZ80の使い方では「n番のI/OポートにAレジスタの内容を出力する」となるのですが，このときには，

アドレス・バス下位　A_7～A_0 ← n
アドレス・バス上位　A_{15}～A_8 ← Aレジスタ

というI/Oアドレス指定がなされます。次の命令をご覧ください。

```
LD  A, 41H
OUT (32H), A
```

この命令列は，X1シリーズにおいては，I/Oポート4132H番地に，データ41Hを出力する命令となります。

6.2 IN命令

今度は入力命令を見ます。

IN r,(C)　…rはA，B，C，D，E，H，Lの各レジスタ。
IN A,(n)　…nは1バイト値。

OUT命令とは逆に，データの流れはI/OポートからCPUのレジスタへとなっていますが，I/Oアドレス指定についての考え方は**OUT**命令のときと全く同様です。

理解度の確認も兼ねて次の例を考えてください。

```
LD  B, 1AH
LD  C, 01H
IN  A,(C)
```

この例では，何番地のI/Oポートの内容がAレジスタに読み出されるでしょうか？　正解はI/Oアドレス1A01H番地です。

今度は，少し難しいかもしれません。今，I/Oポート1900H番地の内容が0DHであるとして，次の命令列を実行後のAレジスタの内容をお考えください。

```
LD  A, 19H
IN  A,(00H)
```

いかがですか？　正解は0DHです。この例ではI/Oポート1900H番地がアクセスされて，その内容0DHをAレジスタに読み込むからです。

IN B,(C)やIN A,(n)のような命令では，実行前のI/Oアドレス指定に用いられたBレジスタやAレジスタの内容が，実行後に変化する可能性がありますから注意しましょう。

6.3 テキストVRAM

I/Oポートには，X1やturboシリーズの機能をひき出すための様々な入出力装置や周辺LSIたちが接続されています。これはメーカーが設計した時点でI/Oポートに割り当てたものですから，私たちユーザーはそのアドレス配置に従って，I/Oポートをアクセスする必要があります。ですから入出力命令の使用にあたっては，X1の**ハードウェア**(機械的しくみ)について理解していることが不可欠です。しかし，これらについて詳細を述べるとなると，それだけで一冊の本になってしまいますから別書に譲って，ここでは**テキスト画面**（英数字・記号・カナなどをふつうに表示する画面）の制御に的を絞ることにしましょう。

画面に文字や図形を表示するには，画面表示専用のメモリーである **Video RAM**（ビデオ・ラム，略して **VRAM** といいます。RAM は Random Access Memory の略で，読み書き自由のメモリーのことです。）に表示したい文字やパターンのコードを書き込みます。X1 シリーズの場合，**VRAM** には文字を表示するための**テキスト VRAM** と，図形パターンを表示するための**グラフィック VRAM**（**GRAM** ともいう）の 2 種類が設けられています。ここではテキスト **VRAM** について考えます。

X1 シリーズでは，テキスト **VRAM** は I/O ポートの 3000H 番地～37FFH 番地に割りあてられています。**VRAM** を I/O ポートに割りつけたため，メインメモリーはすべて自由に使うことができるようになったのが，X1 シリーズの設計の 1 つの特徴です。

さて，テキスト **VRAM** の各 I/O アドレスはテキスト画面上の各表示位置と対応します。たとえば，次の BASIC プログラムでの20，12というのが表示位置を示す座標で，20は x 座標，12は y 座標といいます。

```
10  LOCATE  20, 12
20  PRINT  "A"
30  END
```

テキスト画面では横 1 行に何文字表示するかを**桁数（カラム＝column）**といって，X1 シリーズでは40桁と80桁の 2 通りが指定できます。**WIDTH 40**とすると40桁画面，**WIDTH 80**とすると80桁画面になります。以下の説明は40桁画面で統一したいと思いますので，もし80桁画面になっている方は **WIDTH 40**を実行しておいてください。

テキスト画面の縦方向に注目しましょう。ここでは何行分を表示できるかが大切です。従来の X1 シリーズでは，縦方向の行数は25と決められていましたが，turbo シリーズではこれ以外の行数(10，12，20)が可能になっています。ただ，本書では従来の X1 シリーズのユーザーの方と一緒に読んでいただくために，行数は25行でいきたいと思います。ですから，turbo の方は必要なら，**WIDTH 40,25：KEYLIST 0：CONSOLE 0,25**を実行しておくとよいでしょ

う（あとの2つのコマンドは，turbo画面下のファンクション・キーの表示を消し，25行全部を使えるようにするためです）。

以上で，全部の皆さんが同一条件で実験をする準備が整ったはずです。このとき，テキストの画面座標はx座標（桁の位置）が0から39まで，y座標（行の位置）が0から24まで可能で，次の関係になっています。

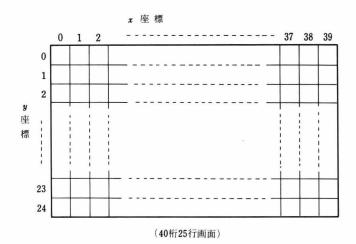



図6.1　テキスト画面の画面座標

（40桁25行画面）

たとえば，**LOCATE 20,12**で指定される位置は，ほぼ画面中央に相当しているわけですね。

さて，このような各表示位置に対応するI/Oアドレスは，次式で求めることができます。

I/Oアドレス＝3000H＋x＋y ＊40

BASICプログラムにしてみると，次のようになります。

```
LIST
10 INPUT "X,Y =";X,Y
20 V=&H3000+X+Y*40
30 PRINT "VRAM=";HEX$(V)+"H"
40 GOTO 10
Ok
RUN
X,Y =? 20,12
VRAM=31F4H
X,Y =? 
```

図6.2 VRAMアドレスの計算

こうして，$x=20$，$y=12$の位置は**VRAM**アドレス31F4H番地で表わされることになりますね。まとめると次のようになっています。

3000H	3001H	3002H	3003H	3004H	3005H	3006H				3025H	3026H	3027H
3028H	3029H	302AH	302BH	302CH	302DH	302EH				304DH	304EH	304FH
3050H	3051H	3052H	3053H	3054H	3055H	3056H				3075H	3076H	3077H
3078H	3079H	307AH	307BH	307CH	307DH	307EH				309DH	309EH	309FH
30A0H	30A1H	30A2H	30A3H	30A4H	30A5H	30A6H				30C5H	30C6H	30C7H
30C8H	30C9H	30CAH	30CBH	30CCH	30CDH	30CEH				30EDH	30EEH	30EFH
30F0H	30F1H	30F2H	30F3H	30F4H	30F5H	30F6H				3115H	3116H	3117H
3370H	3371H	3372H	3373H	3374H	3375H	3376H				3395H	3396H	3397H
3398H	3399H	339AH	339BH	339CH	339DH	339EH				33BDH	33BEH	33BFH
33C0H	33C1H	33C2H	33C3H	33C4H	33C5H	33C6H				33E5H	33E6H	33E7H

図6.3 テキスト**VRAM**アドレスと表示位置

(40桁25行画面)

テキスト**VRAM**に割りあてられている3000H～37FFH番地の約半分しか使用していませんね。あと半分はどうするのでしょう？　答は，40桁画面では2枚の画面（ページ0，ページ1とよぶ）が使えるからです。このうちどちらを表示するかは，**SCREEN**文により決められます。**SCREEN 0,0**と指定するとページ0が表示されますが，**SCREEN 1,1**とするとページ1の方に切り替わります。テキスト**VRAM**アドレスの後半は，ページ1の方に対応していて，ページ0の各**VRAM**アドレスに400Hを加えたものがそうです。たとえば，$x=$

20, $y=12$の位置はページ1では35F4H番地の**VRAM**アドレスを持つわけです。

［注］ 80桁画面にすると，ページは1枚しか持てず，I/Oアドレス＝3000H＋x＋y ∗80の換算式により**VRAM**のアドレスが求められます。

6.4　テキスト属性（アトリビュート）

さあ，画面の表示位置とテキスト**VRAM**の対応がわかったところで，さっそく実験です。画面中央の位置（**VRAM**アドレス＝31F4H）に文字**A**のキャラクタ・コード41Hを出力してみましょう。

アドレス	マシンコード	ニーモニック
D000	01 F4 31	LD BC, 31F4H
D003	3E 41	LD A, 41H
D005	ED 79	OUT (C), A
D007	C9	RET

図6.4　テキスト**VRAM**への出力

```
MON
*M D000
:D000=01 F4 31
:D003=3E 41
:D005=ED 79
:D007=C9
:D008=00
*G D000
*■
```

図6.5　オブジェクトの実行

うまく行きましたか？　文字**A**がキチンと表示された方がおられたら，大変に運がよいといえます。BASICに戻り**CLS**を実行して画面クリア後，もう一

度 D000H 番地からを実行してみてください。今度は妙なパターンが画面中央に表示されるはずです。何故でしょう？

実は，上のプログラムは誤動作をしたのではなく，文字 **A** は表示されているのです。妙なパターンは大きな文字 **A** の一部です。文字 **A**（と思われるパターン）のある行にカーソルを移動し，何かの文字をキーボードより打ってみると，「あら不思議」キチンと **A** に変身しました。ますます，わからなくなりましたか？

この問題を解明するには，テキスト属性について知る必要があります。HuBASIC には，文字の色を指定する **COLOR** 文や，点滅させる **CFLASH**，色の反転をする **CREV**，ユーザー定義文字を表示する **CGEN**，倍文字表示をする **CSIZE** などの命令が設けられていますが，これらはすべて文字をどのように表示するかを決めるもので，**テキスト属性（アトリビュート）**とよばれます。

テキスト属性は１バイトのコード（**属性コード**）で指定され，１バイトの各ビットは次の意味を持っています。

HV	VS	
0	0	標準
0	1	垂直２倍
1	0	水平２倍
1	1	垂直水平２倍

G	R	B	
0	0	0	黒(透明)
0	0	1	青
0	1	0	赤
0	1	1	マゼンタ
1	0	0	緑
1	0	1	シアン
1	1	0	黄
1	1	1	白

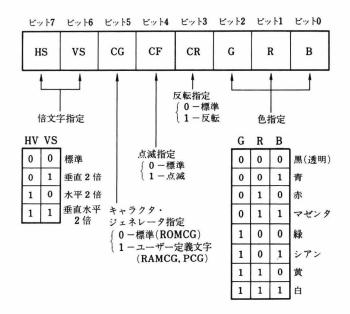



図6.6 テキスト属性コードの各ビットの意味

こうした1バイトデータをI/Oポートに割りつけられている**属性エリア**（**属性ポート，属性VRAM**）に出力することで，テキスト画面の文字の属性が決められます。

属性エリアはI/Oポート2000H番地〜27FFH番地に割りあてられていて，テキスト画面の各表示位置と1対1に対応しています。

```
3000H  <—>  2000H
3001H  <—>  2001H
3002H  <—>  2002H
3003H  <—>  2003H
3004H  <—>  2004H
3005H  <—>  2005H
3006H  <—>  2006H
3007H  <—>  2007H
.....       .....
.....       .....
.....       .....
37FCH  <—>  27FCH
37FDH  <—>  27FDH
37FEH  <—>  27FEH
37FFH  <—>  27FFH
```

図6.7 テキストVRAMと属性エリアのI/Oアドレスの対応

したがって，画面中央（VRAMアドレス＝31F4H）に文字Aを標準白色で表示するには，属性コード07Hを対応する属性エリアであるI/Oポート21F4H番地に出力しなければならないのです。

アドレス	マシンコード	ニーモニック
D000	01 F4 31	LD BC, 31F4H
D003	3E 41	LD A, 41H
D005	ED 79	OUT (C), A
D007	01 F4 21	LD BC, 21F4H
D00A	3E 07	LD A, 07H
D00C	ED 79	OUT (C), A
D00E	C9	RET

図6.8 文字Aを画面中央に標準白色で表示するプログラム

```
MON
*M D000
:D000=01 F4 31
:D003=3E 41
:D005=ED 79
:D007=01 F4 21
:D00A=3E 07
:D00C=ED 79
:D00E=C9
:D00F=00
*G D000
*
                              A
```

図6.9 オブジェクトの実行

さて，ここまで理解できましたら，第1章1.14節で勉強した内容を見直してください。今の皆さんは，そこに登場するマシン語プログラムが画面中央に文字**A**を赤色で表示するものであることを，はっきりと納得できるはずです。皆さんのマシン語の実力は，もうかなりのものなのですよ。自信を持ってください！

6.5 IN 命令で謎を解明！

そうそう，1つ宿題を忘れていました。属性指定をしなかったとき，文字**A**が

アドレス	マシンコード	ニーモニック
D000	01 F4 31	LD BC, 31F4H
D003	3E 41	LD A, 41H
D005	ED 79	OUT (C), A
D007	01 F4 21	LD BC, 21F4H
D00A	ED 78	IN A, (C)
D00C	32 20 D0	LD (0D020H), A
D00F	C9	RET

図6.10 IN 命令の使い方

変に表示されたのは何故か？ という問題に答えておきましょう。そのためには，Aが表示されているときの属性コードを読みとればよいのですね。I/Oポートの属性エリアを読むには，IN命令を使います。図6.10のプログラムが答です。

属性コード1バイトは，D020H番地に格納しておきました。オブジェクトを書き込んだら，BASICへ戻り，CLSを実行しておいてください。

```
MON
*M D000
:D000=01 F4 31
:D003=3E 41
:D005=ED 79
:D007=01 F4 21
:D00A=ED 78
:D00C=32 20 D0
:D00F=C9
:D010=00
*R
Ok
CLS
```

図6.11 オブジェクトの書き込み

再びモニターに入り，G D000でプログラムを実行します。画面中央に変な文字が出たはずです。次に，その属性コードを見るため，＊D D000でダンプします。

```
Ok
MON
*G D000
*D D000
:D000=01 F4 31 3E 41 ED 79 01 /.ホ1>A▞y.
:D008=F4 21 ED 78 32 20 D0 C9 /ホ!▞x2 ミノ
:D010=00 00 00 00 00 00 00 00 /........
:D018=00 00 00 00 00 00 00 00 /........
:D020=47 00 00 00 00 00 00 00 /G.......
:D028=00 00 00 00 00 00 00 00 /........
:D030=00 00 00 00 00 00 00 00 /........
:D038=00 00 00 00 00 00 00 00 /........
:D040=00 00 00 00 00 00 00 00 /........
:D048=00 00 00 00 00 00 00 00 /........
:D050=00 00 00 00 00 00 00 00 /........
:D058=00 00 00 00 00 00 00 00 /........
:D060=00 00 00 00 00 00 00 00 /........
:D068=00 00 00 00 00 00 00 00 /........
:D070=00 00 00 00 00 00 00 00 /........
:D078=00 00 00 00 00 00 00 00 /........
*█
```

図6.12 謎の解明

いかがですか？　属性コードは 47H ですね。垂直 2 倍のビット (VS) が 1 になっているのに注目しましょう。謎は解けましたね。変な文字は **A** を垂直 2 倍にしたときの上半分だったのです。

第7章　桁ずらし，回転，ビット操作

本章で扱う命令は，レジスタやメモリーのビット列あるいは個々のビットに対するものです。

最初に登場するのは，**シフト命令**といって，ビット列を左あるいは右に移動させる命令です。これらは演算としては，2進数の桁ずらしとも考えられ，2進数の2倍（左シフト），1/2倍（右シフト）に対応します。

次に考えるのは，**ローテイト命令**とよばれ，ビット列の最初と最後をつないだ「輪」を左あるいは右に回転させる命令です。この命令を利用して，2進数のかけ算を「筆算形式」で行うプログラムが作れます。

最後に登場するのは，個々のビットを1にしたり（**セット**），0にしたり（**リセット**），1か0かを調べたり（**ビット・テスト**）する命令です。本章では，これらを利用して，「1文字表示プログラム」を汎用サブルーチン化してみます。

7.1　桁ずらし（シフト）

10進数123を10倍すると1230になることは誰でも知っていますが，多くの方は「0を1つ後ろにつける」と理解しておられると思います。ここで，ほんの少し視点を変えてみましょう。

① 123を左へ1桁ずらす。

123 ☐

② 1の位が空になるので，そこに0を入れる。

123 0

「何を面倒な」と思われるかもしれませんが，このように見るだけで皆さんはコンピュータの**シフト命令**を理解する一歩を踏み出したことになるのです。

今述べたこととの対比で，次の例をお考え下さい。

① 2進数1101を左へ1桁ずらす。

1101 [ ]

② 最下位ビットが空になるので，そこに0を入れる。

1101 [0]

さて，この操作は何をしたことになるでしょうか？

答。2進数1101を「2倍」にすることです！ 10進数では1桁左へずらすことは10倍でしたが，2進数では1ビット左へずらすのは2倍に対応します。この操作を「左へ1ビットシフトする」といいます。**shift**（シフト）は「移す，場所を変える」などを意味する英語です。

7.2 左シフト命令

SLA [オペランド] というニーモニックで表わされる命令が，**1ビットの左シフト**をするものです。オペランドは1個で，A，B，C，D，E，H，Lの各レジスタと，(HL)，(IX+d)，(IY+d) で表されるメモリー内容とが許されます。OP コードの **SLA** は **Shift Left Arithmetic** の省略形です。arithmetic は「算術的」と訳され，シフトしてもオーバーフローしない限り「符号つき2進数」として符号が変化しないことを意味します。具体的には，**SLA**命令はオペランドに対して次の働きをします。

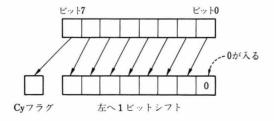



図7.1　SLA 命令の働き

では，**SLA** 命令の具体的な使い方を説明します。たとえば，A レジスタの内容を10倍することを考えましょう。もちろん，加算を10回繰り返してもよいのですが，ここでは **SLA** 命令を使ってみます。シフトで考えるときは，2倍，4倍，8倍，…と倍々の数を組み合わせで考えるのが定石です。10 = 2 + 8 ですから，1ビット左シフト（2倍）と3ビット左シフト（2^3 = 8倍）の加算とすればよいですね。次のプログラム例をご覧ください。

アドレス	マシンコード	ニーモニック
D000	3E 0B	LD A, 11
D002	21 20 D0	LD HL, 0D020H
D005	CB 27	SLA A
D007	77	LD (HL), A
D008	CB 27	SLA A
D00A	CB 27	SLA A
D00C	86	ADD A, (HL)
D00D	77	LD (HL), A
D00E	C9	RET

図7.2　SLA 命令の使い方

この例では，A レジスタの内容 0BH（= 10進の11）を10倍したものを D020H 番地に格納しています。結果は 6EH = 10進数110となって正解です（図7.3）。

原理はおわかりですか？　**SLA A** 命令を合計3回実行しましたね。1回目で A レジスタの内容は2倍されます。それを(HL)に格納します。続けて **SLA A** を2回行うと A レジスタは4倍，8倍されますね。**ADD A, (HL)** で8倍＋2倍 = 10倍ができるわけです。

```
MON
*M D000
:D000=3E 0B
:D002=21 20 D0
:D005=CB 27
:D007=77
:D008=CB 27
:D00A=CB 27
:D00C=86
:D00D=77
:D00E=C9
:D00F=00
*G D000
*D D020 D020
:D020=6E 00 00 00 00 00 00 00 /n.......
*R
Ok
? &H6E
 110
Ok
█
```

図7.3 オブジェクトの実行結果

ただし，注意が必要です。最上位ビットは，つねにCyフラグの方へ移ってしまいますから，少し大きな数を10倍するときはCyフラグによる「桁上がり」を考慮しておかないと正しい計算ができなくなります。ローテイト命令のところ(7.6節) で「大きな数の10倍」をする方法をお見せしますから，もう少しお待ちください。

7.3 右シフト命令

次に右へのシフトについて考えます。左シフトが2倍に対応しますから，右シフトは1/2倍に相当することはよろしいですか？　これも10進数を右へ1桁ずらすと(小数点以下も考えて)1/10倍されることの連想で理解してください。

右シフト命令には**算術的シフト** (arithmetic shift) と**論理的シフト** (logical shift) の2種類があります。まず，算術的シフトを説明します。

算術的右シフトは， SRA オペランド のニーモニックで表記される命令 (Shift Right Arithmetic)で，オペランドはSLA命令と共通です。この命令は，オペランドに対し次のように働きます。

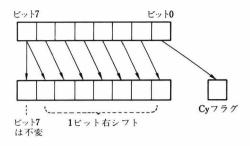



図7.4 SRA 命令の働き

最下位ビット(ビット 0)は Cy フラグへ追い出されますが，最上位ビット(ビット 7) が保たれる点に注目しましょう。符号つき数としての最上位ビットは「符号ビット」でしたね。ですから **SRA** 命令では符号は保存され，これが「算術的」とよぶ理由です。次に **SRA (IX+d)**の例をあげます。

IX＝D400Hで，メモリーのD405H番地の内容がADHであるとき，**SRA (IX+05H)** を実行すると，メモリーのD405H番地とCyフラグの内容は次のようになります．

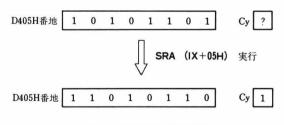



図7.5 SRA (IX+d)の実行例

今度は，**論理的右シフト**を見ます。ニーモニックは **SRL** オペランド で，オペランドは **SRA** 命令と共通です。この命令が「論理的」(**Shift Right Logical**)とよばれるのは， **SRA** と異なり「符号ビット」が保存されず，オペランド内容を単なるビット列と見て右シフトする点です。

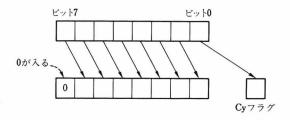



図7.6 SRL 命令の働き

たとえば，Eレジスタの内容が93Hであるとき，**SRL E**を実行するとEレジスタとCyフラグの内容は次のように変化します。

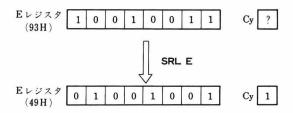



図7.7 SRL Eの実行例

実験プログラムを掲げておきます。

アドレス	マシンコード	ニーモニック
D000	1E 93	LD E, 93H
D002	CB 3B	SRL E
D004	7B	LD A, E
D005	31 20 D0	LD SP, 0D020H
D008	F5	PUSH AF
D009	C3 00 00	JP 0000H

図7.8 実験プログラム

Eレジスタを A レジスタに転送し，F レジスタとともにスタックに **PUSH** して内容を見ます。

```
MON
*M D000
:D000=1E 93
:D002=CB 3B
:D004=7B
:D005=31 20 D0
:D008=F5
:D009=C3 00 00
:D00C=00
*G D000

Ok
MON
*D D000
:D000=1E 93 CB 3B 7B 31 20 D0 /.┬ヒ;(1 ミ
:D008=F5 C3 00 00 00 00 00 00 /火テ......
:D010=00 00 00 00 00 00 00 00 /........
:D018=00 00 00 00 00 00 09 49 /.......I
:D020=00 00 00 00 00 00 00 00 /........
:D028=00 00 00 00 00 00 00 00 /........
:D030=00 00 00 00 00 00 00 00 /........
:D038=00 00 00 00 00 00 00 00 /........
:D040=00 00 00 00 00 00 00 00 /........
:D048=00 00 00 00 00 00 00 00 /........
:D050=00 00 00 00 00 00 00 00 /........
:D058=00 00 00 00 00 00 00 00 /........
:D060=00 00 00 00 00 00 00 00 /........
:D068=00 00 00 00 00 00 00 00 /........
:D070=00 00 00 00 00 00 00 00 /........
:D078=00 00 00 00 00 00 00 00 /........
*■
```

図7.9　結果の確認

7.4　ローテイト命令（ビット回転）

本節で説明する**ローテイト命令**（rotate＝「回転する」の意）は，ビット列を輪のように回転させるものです。初心者の方にとっては，シフト命令より意味をとらえにくいと思いますが，例をたくさん挙げますから，是非マスターしてください。

ローテイト命令は，後述する特別な 4 ビット回転命令を除くと，大きく 4 つに分けられます。

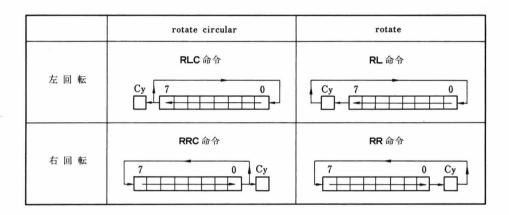



図7.10　ローテイト命令

これらの命令は1個のオペランドを持ち，A，B，C，D，E，H，Lの各レジスタと，(HL)，(IX＋d)，(IY＋d) で表されるメモリー内容がオペランドとして許されます。

ニーモニックのOPコード中に**C**（＝**サーキュラ** circularの略）を持つ命令(**RLC**と**RRC**) では，オペランドの8個のビットが輪のようになって左または右へ回転されます。このとき，ビット7（**RLC**のとき）またはビット0（**RRC**のとき）の内容がCyフラグにコピーされます。

circularでない命令(**RL**と**RR**)では，Cyフラグも含めて9個のビットが輪状をなして回転されます。

たとえば，オペランドがBレジスタで，その内容が7BHであり，現在のCyフラグが1であるとするとき，各ローテイト命令を実行すると，Bレジスタ，Cyフラグの内容は次のように変化します。

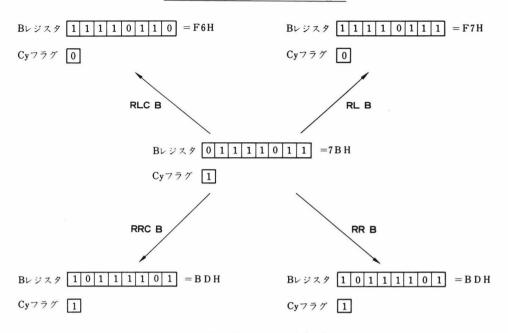



図7.11　Bレジスタにローテイト命令を施したとき

ビットを回転させる操作とは，どんなものか理解できましたか？

Aレジスタを対象とするローテイト命令では，インテル8080 CPUの命令との互換性を持たせるために，特別な1バイト命令が用意されています。

ニーモニック	マシンコード	機　　能
RLCA	07H	RLC　A と同じ．
RLA	17H	RL　A と同じ．
RRCA	0FH	RRC　A と同じ．
RRA	1FH	RR　A と同じ．

図7.12　8080との互換性のための特別な命令

では，ローテイト命令の使い方を例にあたりながら見ていきましょう。

7.5 上位・下位 4 ビットの交換と分離

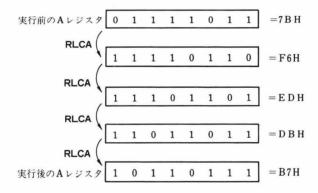
まず簡単なことから。A レジスタの内容が 7BH であるとして，その上位 4 ビット 0111 =7H と下位 4 ビット 1011 =BH とを交換してみましょう。 **RLCA** か **RRCA** を 4 回使えばできますね。今の場合， Cy フラグは特に考えなくて結構です。

アドレス	マシンコード	ニーモニック
D000	3E 7B	LD A, 7BH
D002	07	RLCA
D003	07	RLCA
D004	07	RLCA
D005	07	RLCA
D006	32 20 D0	LD (0D020H), A
D009	C9	RET

図7.13 4 ビット交換

```
MON
*M D000
:D000=3E 7B
:D002=07
:D003=07
:D004=07
:D005=07
:D006=32 20 D0
:D009=C9
:D00A=00
*G D000
*D D000 D020
:D000=3E 7B 07 07 07 07 32 20 />{....2
:D008=D0 C9 00 00 00 00 00 00 /ミ)......
:D010=00 00 00 00 00 00 00 00 /........
:D018=00 00 00 00 00 00 00 00 /........
:D020=B7 00 00 00 00 00 00 00 /キ.......
*■
```

図7.14 オブジェクトの実行と結果の確認

アドレス	マシンコード	ニーモニック
D000	3E 7B	LD A, 7BH
D002	21 20 D0	LD HL, 0D020H
D005	F5	PUSH AF
D006	E6 F0	AND 0F0H
D008	07	RLCA
D009	07	RLCA
D00A	07	RLCA
D00B	07	RLCA
D00C	77	LD (HL), A
D00D	23	INC HL
D00E	F1	POP AF
D00F	E6 0F	AND 0FH
D011	77	LD (HL), A
D012	C9	RET

図7.15 4ビット分離プログラム

これを応用して，AND 命令によるマスク操作と組み合わせると，上位 4 ビットと下位 4 ビットを分離することができます。次の例では，A レジスタの内容 7BH を 4 ビットずつ分け，上位 4 ビットを D020H 番地に，下位 4 ビットを D021H 番地に格納しています(図7.15)。

```
MON
*M D000
:D000=3E 7B
:D002=21 20 D0
:D005=F5
:D006=E6 F0
:D008=07
:D009=07
:D00A=07
:D00B=07
:D00C=77
:D00D=23
:D00E=F1
:D00F=E6 0F
:D011=77
:D012=C9
:D013=00
*G D000
*D D020 D021
:D020=07 0B 00 00 00 00 00 00 /........
*■
```

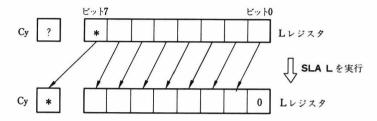
図7.16 オブジェクトの実行と結果の確認

AND 0F0H により A レジスタの下位 4 ビットにマスクをして，上位 4 ビットを残します。もとの A レジスタの内容は壊れてしまうので，あらかじめ **PUSH AF** でスタックへ退避しておきます。**RLCA** を 4 回使って上位 4 ビットを下位 4 ビットに移してメモリーに格納します。次いで **POP AF** でもとの内容に戻し，**AND 0FH** で下位 4 ビットを取り出し，メモリーに入れて終了という仕組みです。よろしいですか？

7.6 16ビット・シフト

8 ビットのシフトにはシフト命令が用意されていました。ではレジスタ・ペアをシフトするにはどうすればよいでしょうか？ 例として，HL レジスタ・

ペアを左にシフトする命令（**SLA HL** とでも呼ぶべきもの）に相当するサブルーチンを作ってみましょう。考え方は以下の通りです（図中＊印で示したLレジスタのビット7の移動に注目しましょう）。

① Lレジスタを左へシフトする．

② RL HによりHレジスタを左回転する．

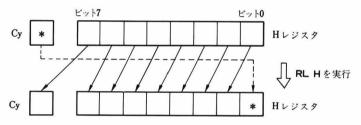



図7.17 仮想命令 SLA HL の考え方

サブルーチンは D100H 番地から配置し，**SHIFT** というラベルをつけておきます。

アドレス	マシンコード	ニーモニック
D100	CB 25	SHIFT : SLA L
D102	CB 14	RL H
D104	C9	RET

図7.18 16ビット・シフトのサブルーチン SHIFT

では，SHIFT サブルーチンを用いて，1バイトの数を10倍するメインルーチンを作ります。1バイト数値は最大でも255ですから，それを10倍しても2550で16ビットに収まります。7.2節で注意した「桁上がり」の問題はこれで解決できますね。今度は，16ビットのアキュムレータとしてHLレジスタ・ペアを，16ビット演算の補助にBCレジスタ・ペアを，またアドレス・ポインタとしてIXレジスタを使うことにします。前に図7.2で挙げたプログラムと全く同じ構造ですから，参照してください。

アドレス	マシンコード	ニーモニック
D000	21 FF 00	LD HL, 255
D003	DD 21 20 D0	LD IX, 0D020H
D007	CD 00 D1	CALL SHIFT
D00A	E5	PUSH HL
D00B	CD 00 D1	CALL SHIFT
D00E	CD 00 D1	CALL SHIFT
D011	C1	POP BC
D012	09	ADD HL, BC
D013	DD 74 00	LD (IX+0), H
D016	DD 75 01	LD (IX+1), L
D019	C9	RET

図7.19 255＊10＝2550の計算

結果は，D020H番地とD021H番地に上位バイト，下位バイトの順に格納されます。答の2550を16進表示すると09F6Hですから，正解ですね！（図7.20）

［注］ HLレジスタ・ペアを左シフトするだけなら ADD HL, HL によっても実現できますが，ここではローテイト命令を用いたCyフラグの取り込み方を覚えていただく

ためにSHIFTサブルーチンの形にしました。また，この考え方を理解しておけば，16ビット右シフトも容易に作ることができますね。

```
MON
*M D000
:D000=21 FF 00
:D003=DD 21 20 D0
:D007=CD 00 D1
:D00A=E5
:D00B=CD 00 D1
:D00E=CD 00 D1
:D011=C1
:D012=09
:D013=DD 74 00
:D016=DD 75 01
:D019=C9
:D01A=00
*M D100
:D100=CB 25
:D102=CB 14
:D104=C9
:D105=00
*G D000
*D D020 D021
:D020=09 F6 00 00 00 00 00 00 /.月......
*■
```

図7.20 オブジェクトの実行と結果の確認

7.7 かけ算プログラム

2進数のかけ算について考えます。たとえば，110101×11101の計算は「筆算」では，次のように行います。

```
         1 1 0 1 0 1   ……10進数では 53
    ×)     1 1 1 0 1   ……10進数では 29
     ─────────────
         1 1 0 1 0 1
     1 1 0 1 0 1
   1 1 0 1 0 1
 1 1 0 1 0 1
─────────────────
1 1 0 0 0 0 0 0 0 0 0 1   ……10進数では 1537
```

図7.21 2進数のかけ算

この計算過程を16ビット・レジスタ2本を使って追ってみると，以下のようになるはずです。図で「加える」が行われるのは，かける数11101のビットを右から順に見て，1のときは「加える」，0のときは「加えない」に対応しています。

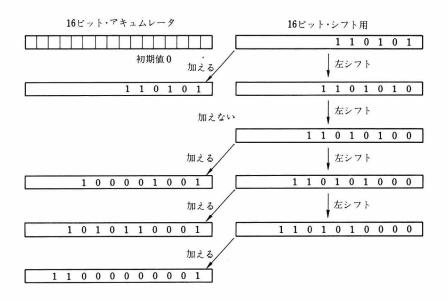



図7.22 16ビット・レジスタを用いて表現してみる

16ビット・アキュムレータにHLレジスタ・ペアをあて，かけられる数をDEレジスタ・ペアに，かける数をAレジスタに入れて流れ図にしたものが図7.23です。

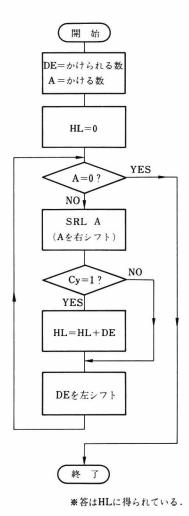



図7.23　2進数かけ算の流れ図

では，この流れ図をもとにプログラムを書いてみましょう。D100H 番地から配置するサブルーチンとし(ラベルは **MULT**)，DE＝**かけられる数**，A＝**かける数**にして呼び出すと，HL＝**答**を入れて返すものとします。

アドレス	マシンコード	ニーモニック
D100	21 00 00	MULT：LD HL, 0
D103	B7	LOOP：OR A
D104	28 0B	JR Z,EXIT
D106	CB 3F	SRL A
D108	30 01	JR NC,SFTDE
D10A	19	ADD HL, DE
D10B	CB 23	SFTDE:SLA E
D10D	CB 12	RL D
D10F	18 F2	JR LOOP
D111	C9	EXIT： RET

※SFTDEは「シフトDE」を意味するラベル．

図7.24　かけ算プログラム

アドレス	マシンコード	ニーモニック
D000	11 35 00	LD DE, 53
D003	3E 1D	LD A, 29
D005	DD 21 20 D0	LD IX, 0D020H
D009	CD 00 D1	CALL MULT
D00C	DD 74 00	LD (IX+0), H
D00F	DD 75 01	LD (IX+1), L
D012	C9	RET

※ 53=35H，29=1DH

図7.25　メイン・ルーチン（53×29を計算）

モニターより，オブジェクトを書き込んでおいてください。

では，実験のためのメイン・プログラムを作ります。図7.21の例 53 × 29 = 1537 をやってみましょう。結果は，D020H 番地に上位バイト，D021H 番地に下位バイトを格納します(図7.25)。

```
*M D000
:D000=11 35 00
:D003=3E 1D
:D005=DD 21 20 D0
:D009=CD 00 D1
:D00C=DD 74 00
:D00F=DD 75 01
:D012=C9
:D013=00
*G D000
*D D020 D021
:D020=06 01 00 00 00 00 00 00 /........
*R
Ok
?&H601
 1537
Ok
█
```

図7.26 実行と確認

正解でしたね！

ローテイト命令の使い方，おわかりになりましたか？

7.8 特殊なローテイト命令

ローテイト命令には，以上で説明したもののほかに，4ビット単位で回転させる特殊な命令 **RLD** および **RRD** があります。これらの命令は，A レジスタおよび HL レジスタ・ペアの示すメモリー内容(HL)との間での回転を行います。

4ビット単位であることからわかるように，2進化10進数（BCD）の演算をするのに適した命令です。

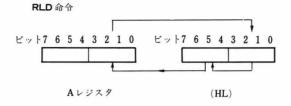
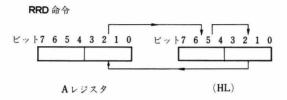



図7.27　4ビット回転

7.9　ビット操作命令

このあと本章で扱う命令は，次の3系列の命令です。

OPコード	第1オペランド	第2オペランド
SET RES BIT のいずれか．	0，1，2，3，4，5，6，7のいずれかで，操作したいビット位置を示す．	・A，B，C，D，E，H，Lの各レジスタ ・(HL)，(IX+d)，(IY+d) の各メモリー内容

図7.28　ビット操作命令

7.10 セット命令とリセット命令

SET 命令は，第2オペランドの特定のビット位置（第1オペランドで示される）を〝1″にする（**セットする**）ものです。RES 命令は，逆に特定のビット位置を〝0″にします（**リセットする**）。

これらの命令はさまざまな場面で使われるでしょうが，X1シリーズにおいては，たとえば次のような使い方をすると便利です。

私たちは第6章6.3節と6.4節において，X1シリーズでのテキスト画面表示の方法について学びました。画面中央に文字 A を標準白色で表示するプログラムをもう一度見直してみましょう（40桁画面にしてください)。

```
LD    BC, 31F4H
LD    A, 41H
OUT   (C), A
LD    BC, 21F4H
LD    A, 07H
OUT   (C), A
RET
```

ここで考えたいのは，テキスト VRAM アドレスと属性エリアのアドレスの対応です。この2つの違いはどこにあるでしょうか。

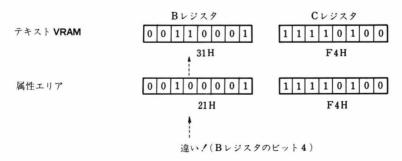



図7.29 テキスト VRAM と属性エリアのアドレス対応

2つのアドレスの違いは，上位8ビットであるBレジスタのビット4が1であるか(テキスト **VRAM**)，0であるか(属性エリア)だけですね。したがって，次のようにして一方から他方へ簡単に直すことができます。

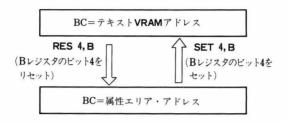



図7.30　テキスト VRAM アドレス↔属性エリア・アドレス

これを用いると，先の**文字A表示プログラム**は次のようにも書けます。

アドレス	マシンコード	ニーモニック
D000	01　F4　31	LD　BC, 31F4H
D003	3E　41	LD　A, 41H
D005	ED　79	OUT (C), A
D007	CB　A0	RES 4, B
D009	3E　07	LD　A, 07H
D00B	ED　79	OUT (C), A
D00D	C9	RET

図7.31　文字A表示プログラム（RES 命令使用）

実行して，同じ結果が得られることを確認してください。

```
MON
*M D000
:D000=01 F4 31
:D003=3E 41
:D005=ED 79
:D007=CB A0
:D009=3E 07
:D00B=ED 79
:D00D=C9
:D00E=00
*G D000
*
```

図7.32 実行と確認

7.11 1文字表示サブルーチン

前節のプログラムは画面中央という特定位置に文字Aだけを表示するものでしたが，1文字表示はテキスト画面制御の基礎ですから，汎用のサブルーチンとして作ってみましょう。

サブルーチン名	1文字表示
入力レジスタ	BC＝テキストVRAMアドレス D＝キャラクタ・コード E＝属性コード（アトリビュート）
壊されるレジスタ	なし．
機能	BCレジスタ・ペアで指定された表示位置に，Dレジスタの指定するキャラクタをEレジスタの属性コードで表示する．

図7.33 1文字表示サブルーチン仕様

サブルーチンはD100H番地から配置しますが（ラベル **CHROUT**），リロケータブルなので任意アドレスから配置できます。

アドレス	マシンコード	ニーモニック
D100	ED 51	CHROUT : OUT (C), D
D102	CB A0	RES 4, B
D104	ED 59	OUT (C), E
D106	CB E0	SET 4, B
D108	C9	RET

図7.34 サブルーチン CHROUT

では，このサブルーチンを使って，画面中央の行に文字Aを標準白色で40個表示してみましょう（中央行は y 座標12で，左端の **VRAM** アドレスは31E0H番地です)。以下のようにメインルーチンを作りました。

アドレス	マシンコード	ニーモニック
D000	21 E0 31	LD HL, 31E0H
D003	16 41	LD D, 41H
D005	1E 07	LD E, 07H
D007	06 28	LD B, 40
D009	C5	LOOP : PUSH BC
D00A	44	LD B, H
D00B	4D	LD C, L
D00C	CD 00 D1	CALL CHROUT
D00F	23	INC HL
D010	C1	POP BC
D011	10 F6	DJNZ LOOP
D013	C9	RET

図7.35 メインルーチン（文字Aを1行表示）

```
MON
*M D000
:D000=21 E0 31
:D003=16 41
:D005=1E 07
:D007=06 28
:D009=C5
:D00A=44
:D00B=4D
:D00C=CD 00 D1
:D00F=23
:D010=C1
:D011=10 F6
:D013=C9
:D014=00
*M D100
:D100=ED 51
:D102=CB A0
:D104=ED 59
:D106=CB E0
:D108=C9
:D109=00
*■
```

図7.36　オブジェクトの書き込み

1行分40回をループさせるために，Bレジスタをカウンタとする条件ジャンプ **DJNZ** 命令を使いましたが，X1シリーズにおいては **DJNZ** の使い方に注意が必要です。というのは，I/Oポートのアクセスにも Bレジスタを使うからです。したがって，本プログラムのようにカウンタとしてのBレジスタを **PUSH BC** と **POP BC** を用いて，ループ内ではスタックに退避させるなどの工夫が必要です。それに関連して，Bレジスタをカウンタとして使っていますから，**VRAM** アドレスのポインタとして HL レジスタ•ペアをあてています。ループ内で HL の内容を BC に転送して，サブルーチン **CHROUT** を呼び出します。表示後，**INC HL** でアドレスを隣に進めます。

では，画面をクリアして実行してみましょう。

```
*G D000
*■




AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

図7.37 実行（1行表示）

成功しましたね！

7.12 ビット・テスト命令

BIT命令は，第2オペランドの特定のビット位置が〝1″か〝0″かを調べ（これを**テスト**といいます），Zフラグを変化させます。SETやRES命令と異なり，第2オペランドの内容を変化させるものではありません。ちょうど，CP命令（比較命令）のようなものだと思えばよいでしょう。

たとえば，次のサブルーチンはI/Oポート1A01H番地のビット6が〝0″になるまで待つためのものです。IN A,(C)でAレジスタにI/Oポートの状態を読みとり，BIT 6,Aによりビット6を調べています。

```
       PUSH BC
       LD   BC, 1A01H
LOOP : IN   A, (C)
       BIT  6, A
       JR   NZ, LOOP
       POP  BC
       RET
```

図7.38 BIT命令の使い方

［注］ このサブルーチンは，X1シリーズのメインCPUから，サブCPUへデータを送り出すときの，タイミングをとるために用いられます。

第8章　交換命令

本章で扱うのは，レジスタ間あるいはレジスタとメモリー（スタック）間での内容の**交換**（入れ換え）をする命令たちです。ロード命令の一種とも考えられますが，「双方向」であってソースとデスティネーションが完全に入れ換わる点が異なります。いずれの命令も，「交換」を意味する英語exchange（エクスチェンジ）にちなんだ**EX**あるいは**EXX**がOPコードになっています。

8.1　DEとHLの交換

EX DE, HLというニーモニックで表わされる命令は，レジスタ・ペア**DE**と**HL**の2バイトの内容を交換するものです。たとえば，DE＝1234H，HL＝5678Hのとき，**EX DE, HL**を実行すると，DE＝5678H，HL＝1234Hとなります。

では，この命令の使用例を説明します。たとえば「**ADD DE, BC**」とでも呼ぶべき命令を行いたいとしましょう。しかし，Z80の命令中にはこれを直接行う命令はありません。

そのようなとき，**ADD HL, BC**を利用して，次のように目的を達することができます。

```
EX   DE, HL
ADD  HL, BC
EX   DE, HL
```

8.2 スタックとの交換

```
EX (SP), HL
EX (SP), IX
EX (SP), IY
```

の3個の命令は，いずれもスタックの内容2バイトと16ビット・レジスタの間での交換を行うものです。ただし，上下位バイトの逆転が生じます。たとえば，EX (SP), HL では

(SP) ↔ H
(SP+1) ↔ L

というように交換されます。ただし，いずれもスタック・ポインタ SP の値自体は変化しません。

この命令の使い方の1つは，スタックを経由することにより16ビット・レジスタ間の勝手な交換が実現できることです。たとえば Z80 にない「EX BC, HL」に相当する命令を実現するには次のようにします。

```
PUSH    BC
EX      (SP), HL
POP     BC
```

少しわかりにくいかもしれませんから，流れを追ってみましょう。

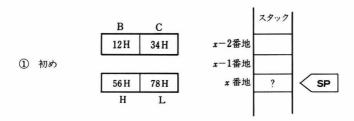
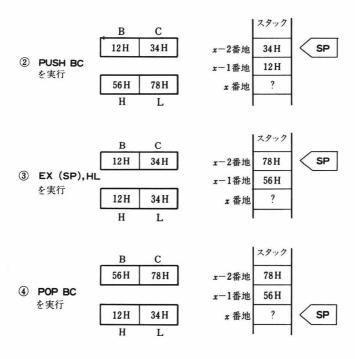



図8.1 仮想命令 **EX BC,HL** の実現

いかがですか？ BCとHLの内容が交換されましたね。

8.3 スタックを用いた文字列表示

スタックとの交換命令のもう1つの面白い使い方を説明します。まずは，図8.2のプログラムをご覧ください。これは何をするプログラムでしょうか？ わかる部分もありますね。D200H番地から始まるサブルーチン**CHROUT**は，すでに学んだ「1文字表示」のサブルーチンです。問題は**MESOUT**とラベルのつけられたD100H番地からのサブルーチンです。

アドレス	マシンコード	ニーモニック
D000	CD 00 D1	CALL MESOUT
D003	54 68 69	DATA： DEFM 'Thi'(注1)
D006	73 20 69	DEFM 's␣i'
D009	73 20 61	DEFM 's␣a'
D00C	20 74 65	DEFM '␣te'
D00F	73 74 2E	DEFM 'st.'
D012	00	DEFB 00H (注2)
D013	C9	GOMON： RET
D100	01 E0 31	MESOUT：LD BC,31E0H
D103	1E 07	LD E, 07H
D105	E3	LOOP： EX (SP), HL
D106	7E	LD A, (HL)
D107	23	INC HL
D108	E3	EX (SP), HL
D109	B7	OR A
D10A	C8	RET Z
D10B	57	LD D, A
D10C	CD 00 D2	CALL CHROUT
D10F	03	INC BC
D110	18 F3	JR LOOP
D200	ED 51	CHROUT：OUT (C), D
D202	CB A0	RET 4, B
D204	ED 59	OUT (C), E
D206	CB E0	SET 4, B
D208	C9	RET

（注1） DEFMは，アセンブリ言語の命令の1つで，メモリー内に文字列(メッセージ)をセットするもの．DEFMに続いて '文字列' と指定すると，文字列内の各文字に対応するキャラクタ・コードがセットされる．

（注2） DEFBも，アセンブリ言語の命令で，メモリー内に1バイト単位のデータをセットするもの．たとえば，DEFB 00Hは，その番地にデータ00Hをセットする．

図8.2 スタック交換命令の面白い使い方

とりあえず実験してみましょう。モニターよりオブジェクトを書き込んでください。

```
MON
*M D000
:D000=CD 00 D1
:D003=54 68 69
:D006=73 20 69
:D009=73 20 61
:D00C=20 74 65
:D00F=73 74 2E
:D012=00
:D013=C9
:D014=00
*M D100
:D100=01 E0 31
:D103=1E 07
:D105=E3
:D106=7E
:D107=23
:D108=E3
:D109=B7
:D10A=C8
:D10B=57
:D10C=CD 00 D2
:D10F=03
:D110=18 F3
:D112=00

*M D200
:D200=ED 51
:D202=CB A0
:D204=ED 59
:D206=CB E0
:D208=C9
:D209=00
*■
```

図8.3　オブジェクトの書き込み

できましたら，一度 BASIC に戻り，**CLS** で画面クリア後，再びモニターに入り，**＊ G D000**で実行してください（図8.4）。

何やら画面中央行に，文字列が表示されましたね。そうです！　**MESOUT** とラベルをつけたサブルーチンは文字列表示をするものであったのです（ラベルは MESsage OUT のつもりです）。その文字列はどこにあるかといえば，メインルーチン中にアセンブリ言語命令 **DEFM** でセットされているわけですね。さて，どのような仕組みで，これが表示されているのでしょう。秘密は **MESOUT**

サブルーチン内の **EX (SP), HL** にありそうですね。

CALL MESOUT が実行されると，「戻り番地」D003H がスタックへ PUSH されてから，D100H 番地へジャンプするのでしたね。では，CPU になったつもりで流れを追っていきましょう(図8.5)。

```
Ok
MON
*G D000
*■




This is a test.
```

図8.4 実　行

① **CALL MESOUT** でサブルーチンに飛び込んだときのスタックの状態．

② **LD BC, 31E0H**
（BCに表示開始 **VRAM** アドレスをセット）

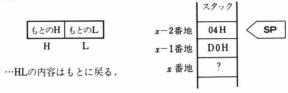
③ **LD E, 07H**
（Eレジスタに標準白色の属性コード07Hをセット）

④ **EX (SP), HL**

⑤ **LD A, (HL)**
…Aレジスタに D003H番地の内容54H（文字Tのコード）が入る.

⑥ **INC HL** ……HL＝D004Hとなる.

⑦ **EX (SP), HL**

…HLの内容はもとに戻る.

⑧ **OR A**
RET Z ｝Aレジスタの内容が00Hならリターン
（今の場合，A＝54Hなので リターンしない.）

⑨ **LD D, A**
（Dレジスタにキャラクタ・コード54Hがセットされる.）

⑩ **CALL CHROUT**
（**VRAM**アドレス31E0Hに文字Tを表示）

⑪ **INC BC**
（**VRAM**アドレスを隣に進めて，BC＝31E1Hとなる.）

⑫ **JR LOOP**
EX (SP), HL

…HL＝D004Hとなる.

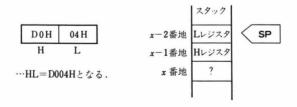



図8.5 サブルーチン **MESOUT** の流れを追う

いかがですか？　D003H 番地から格納されている表示用データがサブルーチン **CHROUT** に送られる仕組みがわかりましたか？　以下，このようなことを繰り返して文字列が表示できるわけですね。

さて，データの最後 D012H 番地には 00H が書かれていますが，これを **MESOUT** 内で読みとると，**OR A** と **RET Z** によって，スタックからプログラム・カウンタ PC に「戻り番地」が **POP** されますが，このときのスタックには D013H が積まれているはずで，したがって D013H 番地に戻ってくることになります。すると，ここには **RET** 命令が書かれているので，めでたくモニターに戻れるのですね。

このようにスタックとの交換命令は，スタックをデータ・アドレスの受け渡しエリアとして用いるのにも使えます。昔，マシン語の勉強をしていて交換命令のこうした使用法を初めて見たとき，「何てカシコイ使い方だろう！」と感心したものでしたが，皆さんはどのように感じられましたか。

8.4　補助レジスタとの交換

Z80 では，A，F，B，C，D，E，H，L の各レジスタについて，全く同じ機能を持つ補助レジスタ A′，F′，B′，C′，D′，E′，H′，L′ が設けられています。普通は，これら補助レジスタを使うことはめったにないのですが，複雑なプログラムで多くのレジスタを必要とするときや，レジスタ内容を保存したいときなどに使うことがあります。ただし，たとえば A レジスタと A′ レジスタが同時に使えるというのではなく，あくまで一方しか使えません。A′ を使いたいときは，まず，A と A′ を「交換」して，以後もとの A′ を新たな A として使う必要があります。このために設けられているのが補助レジスタとの交換命令で，2 つの命令があります。

EX AF, AF′ というニーモニックで表わされる命令は，A レジスタと F レジスタについて補助レジスタとの交換をするものです。これを実行すると，A レジスタと A′ レジスタの内容，F レジスタと F′ レジスタの内容が各々交換されます。

他のレジスタは一斉に交換されます。**EXX** とニーモニック表記される命令がそれで，これを実行すると以下のようになります。

Bレジスタの内容	↔	B′レジスタの内容
C	〃	↔	C′	〃
D	〃	↔	D′	〃
E	〃	↔	E′	〃
H	〃	↔	H′	〃
L	〃	↔	L′	〃

図8.6　EXX 命令の機能

繰り返し強調しますが，補助レジスタが登場するのは，**EX AF, AF′** と **EXX** だけであって，他の命令はすべて「主レジスタ」を対象としていることに注意しましょう。

たとえば，あるサブルーチン中でレジスタ類の内容を保存したいときは，**PUSH** と **POP** でスタックに退避する方法がありますが，それ以外にこうした補助レジスタとの交換を用いることもできます。

```
EX   AF,  AF′  } 補助レジスタとの交換
EXX            }
[サブルーチン内の処理]
EXX            } レジスタをもとに戻す
EX   AF,  AF′  }
RET
```

ただあまり頻繁に使うと，現在のレジスタが「主」か「補助」か自分でもわからなくなりますから，注意しましょう。

第9章　ブロック命令

個々のメモリー番地やI/Oポートの内容ではなく，ある一塊（ブロック）のメモリー内容やI/Oポート内容を対象とした命令を**ブロック命令**とよびます。これらの命令はすべて，8080（Z80の先祖）にはなくZ80で盛り込まれたもので，複数のレジスタが関与する高機能命令になっています。

本章では，**ブロック転送**，**ブロック・サーチ**，**ブロック入出力**の命令を説明します。

9.1　ブロック転送

連続した番地をもつメモリー内容を一塊として，他の番地に転送するのを**ブロック転送**とよびます。これを行うための命令が4つ用意されています。

LDI　(load increment)
LDIR　(load increment repeat)
LDD　(load decrement)
LDDR　(load decrement repeat)

まず，LDIR命令から見ましょう。この命令は，HLレジスタ・ペアの示す番地から始まるメモリー内容を，DEレジスタ・ペアの示す番地以降へ，BCレジスタ・ペアの示すバイト数だけブロック転送する命令です。1バイト分転送するごとに，HLとDEの内容は＋1（increment）され，BCの内容は－1されます。BC＝0となったところで，命令の実行が終了します。

例として，0000H番地～007FH番地の内容128バイト（＝80H）をE000H番地以降にブロック転送してみましょう。HLは転送元のアドレス0000Hを，

DEには転送先のアドレスE000Hを，そしてBCにはバイト数0080Hをセットしてから**LDIR**命令を実行します。

アドレス	マシンコード	ニーモニック
D000	21　00　00	LD　HL，0000H
D003	11　00　E0	LD　DE，0E000H
D006	01　80　00	LD　BC，0080H
D009	ED　B0	LDIR
D00B	C9	RET

図9.1　LDIR命令の使い方

ではオブジェクトを書き込んで実行します。キチンと転送されているか，E000H番地からダンプしてください。

```
MON
*M D000
:D000=21  00  00
:D003=11  00  E0
:D006=01  80  00
:D009=ED  B0
:D00B=C9
:D00C=00
*G D000
*D E000
:E000=C3  FA  00  C3  7C  01  50  28  /テ秒・テ|・P(
:E008=C3  D3  03  C3  83  04  00  09  /テモ・テ■・・・
:E010=C3  D3  03  C3  BC  04  00  18  /テモ・テシ・・・
:E018=C3  D3  03  C3  9D  02  00  27  /テモ・テ'・・'
:E020=C3  D3  03  C3  07  0E  07  20  /テモ・テ・・・
:E028=C3  D3  03  C3  AA  0A  0D  37  /テモ・テェ・・7
:E030=C3  D3  03  C3  CA  0A  00  FF  /テモ・テハ・・〒
:E038=C3  D3  03  C3  75  0B  C3  79  /テモ・テu・テy
:E040=0B  C3  9A  0B  C3  9E  0B  C3  /・テ┌・テ┌・テ
:E048=AE  0B  C3  30  03  C3  88  09  /ョ・テ0・テ| ・
:E050=00  00  46  03  D3  03  D3  03  /・・F・モ・モ・
:E058=D3  03  D3  03  D3  03  D3  03  /モ・モ・モ・モ・
:E060=D3  03  D3  03  D3  03  C3  FA  /モ・モ・モ・テ秒
:E068=00  63  0E  63  0E  86  07  1E  /・c・c・▌・・
:E070=07  63  0E  F1  07  A8  07  F7  /・c・土・ィ・B
:E078=07  14  08  A1  08  1B  07  BF  /・・・・・・・ソ
*■
```

図9.2　実行と確認

ダンプ画面は，従来のX1シリーズのものです。turboシリーズの方は内容に違いがあるはずですが，それはBASICの違いですのでご安心ください。

次に，LDDR命令を見ます。これも原理はLDIRと同様なのですが，1バイト転送するごとに，HLとDEが−1される(decrement)点が異なっています。したがって，LDDRではアドレスの減少する方向に，次々と転送されるのですね。LDIRの例であげたのと同じ結果を得るには，次のように設定してから，LDDR命令を実行する必要があります。

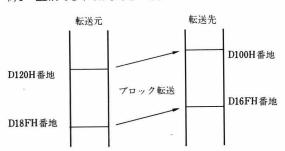
HL＝007FH

DE＝E07FH

BC＝0080H

この例では，LDIRを用いてもLDDRでも結果に変わりはありません。では，2つの命令はどのように使い分けるのでしょう？　答は，転送前と転送後とで，アドレスに重なりのある場合を考えるとわかります。次の例を考えてみましょう。

例1では，転送前と後とでD120H〜D16FH番地が重なっています。この場合はLDIR命令を用いて，前の方から順に転送しないと内容を壊してしまいます。

逆に例2では，LDDR命令で後から前へ向かって順に転送しないと内容が壊れます。

例1　LDIRでなくてはならない場合

例2　**LDDR** でなくてはならない場合

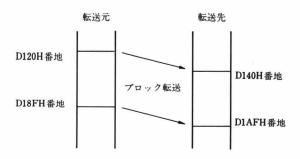



図9.3　LDIR と LDDR の使い分け

以上の2つの命令では，BC＝0となるまで「全自動」で実行されますが，残る2つの命令はいわば「半自動」で，1バイト分転送すると命令の実行を終えます。

LDI 命令では，**LDIR** の最初の1回分の動作が実行されます。すなわち

```
(DE) ← (HL)
 HL  ← HL+1
 DE  ← DE+1
 BC  ← BC-1
```

となります。

同様に，**LDD** 命令は **LDDR** の1回分の動作である

```
(DE) ← (HL)
 HL  ← HL-1
 DE  ← DE-1
 BC  ← BC-1
```

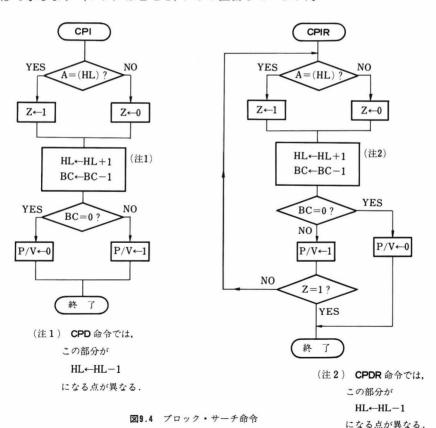
が実行されます。これらの命令は，ブロック転送の1回分ごとに，状態のチェックをしながら行いたいときに使います。

9.2 ブロック・サーチ

A レジスタとメモリー・ブロックの各内容とを対象とした比較命令で，ブロック転送命令と同様に 4 種類あります。

CPI (compare increment)
CPIR (compare increment repeat)
CPD (compare decrement)
CPDR (compare decrement repeat)

これらの命令の実行動作を言葉で説明すると大変ですので，以下に流れ図の形で示します（フラグは Z と P/V のみ注目しています)。

図9.4 ブロック・サーチ命令

いずれの場合も，A レジスタの内容とメモリー内容が一致したか否かはZフラグにより(一致ならZ＝1)，またBC＝0になって終了したか否かはP/Vフラグにより（BC＝0になればP/V＝0）判別されます。

では，感じをつかむために余り実用的ではありませんが，次のプログラムを考えましょう。

アドレス	マシンコード	ニーモニック
D000	31 30 D0	LD SP, 0D030H
D003	21 13 D0	LD HL, DATA
D006	01 04 00	LD BC, 4
D009	3E 03	LD A, 03H
D00B	ED B1	CPIR
D00D	F5	PUSH AF
D00E	C5	PUSH BC
D00F	E5	PUSH HL
D010	C3 00 00	JP 0000H
D013	01	DATA : DEFB 01H
D014	02	DEFB 02H
D015	03	DEFB 03H
D016	04	DEFB 04H

図9.5 CPIRの使用例

このプログラムは4つのデータ01H，02H，03H，04Hの中にAレジスタのデータ03Hを探す（**サーチする**）ものです。HLにはデータの先頭アドレスD013H，BCには比較のバイト数0004Hをセットして**CPIR**命令を実行します。実行後の各レジスタの内容を見るため，スタックを設定して**PUSH**してお

きます。さて，結果を見てみましょう。

```
MON
*M D000
:D000=31  30  D0
:D003=21  13  D0
:D006=01  04  00
:D009=3E  03
:D00B=ED  B1
:D00D=F5
:D00E=C5
:D00F=E5
:D010=C3  00  00
:D013=01
:D014=02
:D015=03
:D016=04
:D017=00
*G D000
```

図9.6 オブジェクトの実行

```
Ok
MON
*D D000
:D000=31  30  D0  21  13  D0  01  04  /10ミ!.ミ..
:D008=00  3E  03  ED  B1  F5  C5  E5  /.>.■アメナ♣
:D010=C3  00  00  01  02  03  04  00  /テ.......
:D018=00  00  00  00  00  00  00  00  /........
:D020=00  00  00  00  00  00  00  00  /........
:D028=00  00  16  D0  01  00  46  03  /...ミ..F.
:D030=00  00  00  00  00  00  00  00  /........
:D038=00  00  00  00  00  00  00  00  /........
:D040=00  00  00  00  00  00  00  00  /........
:D048=00  00  00  00  00  00  00  00  /........
:D050=00  00  00  00  00  00  00  00  /........
:D058=00  00  00  00  00  00  00  00  /........
:D060=00  00  00  00  00  00  00  00  /........
:D068=00  00  00  00  00  00  00  00  /........
:D070=00  00  00  00  00  00  00  00  /........
:D078=00  00  00  00  00  00  00  00  /........
*■
```

（注） 実行後のレジスタ

図9.7 結果の確認

フラグを見ると，Z＝1であり03Hと一致したデータがあったことを示しています。見つかったアドレスよりHLはさらに＋1されていますから，HLの内容であるD016Hより1つ前のD015H番地が，そのデータのある所です。正解ですね。また，P/Vフラグは1ですから，BC≠0で終了したことがわかります。実際，BC＝0001Hであり，まだ，1バイトの比較を残して終了したこともわかります。

D00AH番地の比較データを，いろいろと変えて実験してみてください。

では，もう少し面白いプログラム例を考えます。0000H番地から9FFFH番地までに，C9Hは何回登場するかを数えるプログラム例です。次をご覧ください。

アドレス	マシンコード	ニーモニック
D000	21 00 00	LD HL, 0000H
D003	01 00 A0	LD BC,0A000H
D006	3E C9	LD A, 0C9H
D008	11 00 00	LD DE, 0000H
D00B	ED A1	LOOP：CPI
D00D	28 05	JR Z, CNT
D00F	E2 17 D0	JP PO, EXIT
D012	18 F7	JR LOOP
D014	13	CNT：INT DE
D015	18 F4	JR LOOP
D017	21 30 D0	EXIT：LD HL,0D030H
D01A	72	LD (HL), D
D01B	23	INC HL
D01C	73	LD (HL), E
D01D	C9	RET

図9.8 CPI使用例（RET命令の数）

プログラム中，HLは比較のためのアドレス・ポインタ，BCは比較の残りバイト数，Aレジスタはデータ C9H に使われています。C9H と一致するデータが見つかると，カウンタ DE が＋1されて個数を数えるようにしています。9FFFH 番地までの比較が終わると，BC＝0でP/Vフラグが0になるので **JP PO, EXIT** によりループから脱けて，D030H 番地にカウンタの上位バイト，D031H 番地にカウンタの下位バイトを格納して終了します。オブジェクトを書き込み実行してみてください。

```
MON
*M D000
:D000=21 00 00
:D003=01 00 A0
:D006=3E C9
:D008=11 00 00
:D00B=ED A1
:D00D=28 05
:D00F=E2 17 D0
:D012=18 F7
:D014=13
:D015=18 F4
:D017=21 30 D0
:D01A=72
:D01B=23
:D01C=73
:D01D=C9
:D01E=00
*G D000
*■
```

図9.9 実　行

結果は，X1のディスク BASIC である CZ-8FB01 では次のようになり，C9H は 9FFFH 番地までに572個あることがわかりました。

```
*D D000 D038
:D000=21 00 00 01 00 A0 3E C9 /!....  >ﾉ
:D008=11 00 00 ED A1 28 05 E2 /...▞.(.♠
:D010=17 D0 18 F7 13 18 F4 21 /.ﾐ.日..ﾎ!
:D018=30 D0 72 23 73 C9 00 00 /0ﾐr#sﾉ..
:D020=00 00 00 00 00 00 00 00 /........
:D028=00 00 00 00 00 00 00 00 /........
:D030=02 3C 00 00 00 00 00 00 /.<......
:D038=00 00 00 00 00 00 00 00 /........
*R
Ok
?&H23C
 572
Ok
■
```

図9.10 結果（CZ-8FB01の場合）

他のBASICでも全く同じプログラムで実験したところ，X1のテープBASIC（CZ-8CB01）とturbo BASIC（CZ-8FB02）では同じ結果で551個になりました。

9.3 ブロック入出力

I/Oポートをアクセスする入出力命令に関しても，ブロック命令があります。入力関係4個，出力関係4個の命令が設けられています。

ブロック入力

INI	(in increment)
INIR	(in increment repeat)
IND	(in decrement)
INDR	(in decrement repeat)

ブロック出力

OUTI	(out increment)
OTIR	(out increment repeat)
OUTD	(out decrement)
OTDR	(out decrement repeat)

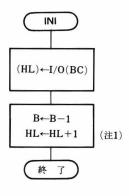
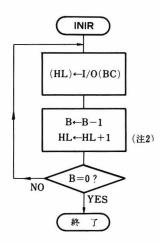
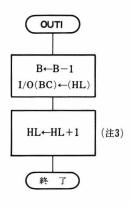
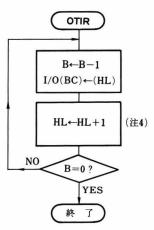
これらの命令の実行動作を流れ図にすると，次のようになります。図中，I/O（BC）は，BCレジスタ・ペアを指定されるアドレスのI/Oポートを示します。Z80の通常の使い方では，I/Oポートのアドレス上位8ビットは無視するので，I/O（C）と書いても同じなのですが，X1シリーズでは周知のようにI/Oポートは16ビットでアドレス指定しますので，敢えてI/O（BC）と表記します。

（注１）　**IND** 命令では，
この部分が
HL←HL－1
となる．

（注２）　**INDR** 命令では，
この部分が
HL←HL－1
となる．

（注３）　**OUTD** 命令では，
この部分が
HL←HL－1
となる．

（注４）　**OTDR** 命令では，
この部分が
HL←HL－1
となる．

図9.11　ブロック入出力命令

さて，これらの命令もブロック転送やブロック・サーチと同様に…としようとすると失敗します。いずれもBレジスタを8ビットのカウンタとする命令ですが，X1シリーズにおいては，上述のようにBレジスタはI/Oアドレス指定での上位8ビットも受け持っていますから，面倒なことが起こります。たとえば，

```
LD   BC, 3000H
LD   HL, 0E000H
OTIR
```

を実行したとすると，X1シリーズにおいては次の動作となります。

```
I/O (2F00H) ← (E000H)
I/O (2E00H) ← (E001H)
I/O (2D00H) ← (E002H)
        ⋮
I/O (0200H) ← (E02DH)
I/O (0100H) ← (E02EH)
I/O (0000H) ← (E02FH)
```

I/Oポートの1＊＊＊H番地のあたりにはシステム用の様々な周辺LSIが接続されているので，上の動作を実行するとまず確実に変なことになります。というわけで，ブロック入出力に関しては，「全自動」（リピートつき）命令である **INIR, INDR, OTIR, OTDR** の4個は **X1シリーズで使わぬ方がよい**命令であるといえます。

しかしながら，「半自動」（リピートなし）の残り4個のブロック入出力命令は，レジスタの動きに注意すればX1シリーズでも利用価値があります。

たとえば，メモリー上E000H番地からE3E7番地までにあらかじめ格納しておいた1000（＝3E8H）バイト分の表示データを，標準白色の属性コードでテキスト画面1画面分（**VRAM** アドレス3000H～33E7H）にブロック出力するには，以下のように **OUTI** 命令を使えます。

アドレス	マシンコード	ニーモニック
D000	21 00 E0	LD HL, 0E000H
D003	01 00 30	LD BC, 3000H
D006	11 E8 03	LD DE, 03E8H
D009	04	LOOP： INC B
D00A	ED A3	OUTI
D00C	CD 16 D0	CALL WHITE
D00F	03	INC BC
D010	1B	DEC DE
D011	7A	LD A, D
D012	B3	OR E
D013	20 F4	JR NZ,LOOP
D015	C9	RET
D016	3E 07	WHITE：LD A, 07H
D018	CB A0	RES 4, B
D01A	ED 79	OUT (C), A
D01C	CB E0	SET 4, B
D01E	C9	RET

図9.12 OUTI命令の使用例

WHITEとラベルをつけたサブルーチンは，属性エリアに白色コード07Hを出力するものですが，本プログラムが（ゲームなどの）もっと大きなプログラムに組み込まれていて，画面の属性設定がすでになされている場合は考慮しなくても構いません。ここでは単体で実行させるため，画面の状態がどうなっているか不明なので，念のため属性を出力しているにすぎません。

本プログラムでは，HLはメモリーのアドレス・ポインタ，BCはVRAMのアドレス・ポインタ，DEはカウンタとして使われています。注意してほしいのは，OUTI命令を実行する前のINC Bです。これはOUTI命令内部で最初にBレジスタの内容が－1されてしまうので(図9.11参照)，前もって＋1しておき，I/Oアドレス指定でのBレジスタが変わらぬようにする配慮です。表示後は，INC BCでVRAMアドレスを進め（HLの方は自動的に＋1されている！），DEC DEでカウンタを減じて，DE≠0である限りループさせます。

```
MON
*M D000
:D000=21 00 E0
:D003=01 00 30
:D006=11 E8 03
:D009=04
:D00A=ED A3
:D00C=CD 16 D0
:D00F=03
:D010=1B
:D011=7A
:D012=B3
:D013=20 F4
:D015=C9
:D016=3E 07
:D018=CB A0
:D01A=ED 79
:D01C=CB E0
:D01E=C9
:D01F=00
*
```

図9.13 オブジェクトの書き込み

実行前に，E000H～E3E7H番地に表示用データを設定しておくのを忘れないでください。

ここでは，次のように設定してみました。

```
*D E1A8 E23F
:E1A8=00 00 00 00 00 00 00 00 /........
:E1B0=00 00 00 00 00 00 00 00 /........
:E1B8=20 20 20 20 20 20 20 20 /
:E1C0=20 20 20 20 20 20 20 20 /
:E1C8=20 20 20 20 20 20 20 20 /
:E1D0=20 20 20 20 20 20 20 20 /
:E1D8=20 20 20 20 20 20 20 20 /
:E1E0=BA DA CA A4 20 4F 55 54 /ｺﾚﾊ､ OUT
:E1E8=49 D2 B2 DA B2 C3 DE A4 /Iﾒｲﾚｲﾃﾞ､
:E1F0=20 CB AE B3 BC DE BC C0 / ﾋｮｳｼﾞｼﾀ
:E1F8=20 B6 DE D2 DD C3 DE BD / ｶﾞﾒﾝﾃﾞｽ
:E200=A1 20 20 20 20 20 20 20 /｡
:E208=20 20 20 20 20 20 20 20 /
:E210=20 20 20 20 20 20 20 20 /
:E218=20 20 20 20 20 20 20 20 /
:E220=20 20 20 20 20 20 20 20 /
:E228=20 20 20 20 20 20 20 20 /
:E230=00 00 00 00 00 00 00 00 /........
:E238=00 00 00 00 00 00 00 00 /........
*■
```

図9.14 表示データの設定

さて，＊ **G D000**で実行すると，画面は図のようになります。

```
ｺﾚﾊ､ OUTIﾒｲﾚｲﾃﾞ､ ﾋｮｳｼﾞｼﾀ ｶﾞﾒﾝﾃﾞｽ｡

*■
```

（注） ＊■は実行後のカーソルで，必ずしもこの位置には出ません．

図9.15 実　　行

画面の大部分を占める妙なマークは，00H をキャラクタ・コードにもつキャラクタです。

第10章　CPU制御命令

CPUは，普通の状態では，

①メモリーから命令を読み出す，

②命令を解読する，

③命令を実行する，

④再び①に戻る，

という動作を続けていますが，本章では，このようなCPUの動作そのものを制御する命令たち(**NOP**命令，**HALT**命令，割り込み関係の命令)が登場します。

本章で，Z80 CPUの命令の解説は完結しますが，本章の命令は初めての方には少し難しいかもしれません(とくに，割り込みについて)。実際，マシン語学習の初期には，**NOP**を除いてCPU制御命令は，ほとんど使うことはないでしょうから，**NOP**の使い方(10.1節から10.3節まで)をマスターすることから始めるとよいと思います。

10.1　NOP命令

No operation（**ノー・オペレーション**）の省略形**NOP**をニーモニックにもつ命令で，文字通り「何もしない」命令です。マシンコードは00Hです。

厳密に言うと，「何もしない」というのは誤りで，プログラム・カウンタ(PC)を次の命令まで1だけ進め，メモリーをリフレッシュする信号を出します。

［注］ メモリー用のIC（集積回路）には，大きく分けて**ROM**（Read Only Memory＝読み出しのみ可能）と**RAM**（Random Access Memory＝読み書きともに可能）の2つがあります。RAMにはさらに**スタティックRAM**（電源を切るか，別のデータを書かぬ限りもとの内容を保持する）と**ダイナミックRAM**（つねにリフレッシュしてないと内容が保持されない）という区別があります。X1シリーズのメイン・メモリーにはダイナミックRAMが使われているので，つねにCPUからリフ

レッシュ信号を出して内容を保持しておかねばならないのです。Z80では各命令実行時にあるタイミングで，リフレッシュ信号が出されますが，**NOP**命令のときも，リフレッシュ信号はキチンと出され，ダイナミックRAMの記憶内容が失なわれないようにしているわけです。

また，X1シリーズのように，Z80 Aを4 MHz（メガ・ヘルツ）で動作させている場合には，**NOP**命令を実行するのに1 μs（マイクロ秒＝100万分の1秒）という時間を要します。

こうした**NOP**命令の働きを理解すると，それなりに応用の場面があることがわかるでしょう。

10.2 デバッグへの利用

まずマシン語プログラムの**デバッグ**（プログラムの誤りである**バグ**をとり除

マシンコード	ニーモニック
01 00 20	LD BC, 2000H
11 E8 03	LD DE, 1000
ED 78	LOOP：IN A, (C)
CB DF	SET 3, A
ED 79	OUT (C), A
03	INC BC
1B	DEC DE
7A	LD A, D
B3	OR E
20 F4	JR NZ, LOOP
C9	RET

図10.1 1画面の白黒を反転するサブルーチン

くこと）のとき，削除した命令の埋め合わせのために使います。たとえば，次のプログラムのオブジェクトを，モニターより書き込んでいるとしましょう(図10.1)。

プログラムの意味については，テキスト画面表示のところで学んだ属性コードを復習してください（第6章6.4節参照。属性コードのビット3を〝1″にすると **CREV1**になります)。

さて，モニターの **M** コマンドで書き込んでいるとき，**つい手がすべって**3バイト目の20Hをダブッて書いてしまったとします。

```
MON
*M D000
:D000=01 00 20 20 11 E8 03 ED
:D008=78 CB DF ED 79 03 1B 7A
:D010=B3 20 F4 C9
:D014=00
*█
```

図10.2 つい手がすべって…。

このプログラムは実行してはいけませんよ！ なぜなら，CPUは実行時に次の意味だと判断してしまうからです(図10.3)。

プログラムの最初の方の解釈が変わってしまって，意味不明な命令列となりましたね。マシン語ではこのようにわずか1バイトの誤りが重大な失敗につながることがよくありますから注意しましょう。

では,正しく修正するにはどうすればよいでしょうか？ BASICプログラムであれば，ある行を削除すればシステムが自動的に行を詰めてくれましたが，マシン語ではそうはいきません。1つの手はモニターの **T** コマンドで5バイト目(11H)以降を1つ上に「ブロック転送」する方法がありますが，大きなプログラムに組み込まれて身うごきできない場合もあります。そんなとき威力を発揮するのが，**NOP** 命令なのです。

余分な4バイト目の20Hを00Hに変更しましょう。

マシンコード	ニーモニック
01　00　20	LD　BC, 2000H
20　11	JR　NZ, +19
E8	意味不明 RET PE
03	INC　BC
ED　78	LOOP：　IN　A, (C)
CB　DF	SET 3, A
ED　79	OUT (C), A
03	INC　BC
1B	DEC DE
7A	LD　A, D
B3	OR　E
20　F4	JR　NZ, LOOP
C9	RET

図10.3　20H を余分に入れたプログラムの意味

```
*D D000 D017
:D000=01 00 20 20 11 E8 03 ED /..   .X.▞
:D008=78 CB DF ED 79 03 1B 7A /xヒ"▞y..z
:D010=B3 20 F4 C9 00 00 00 00 /ウ ホノ....
*M D003
:D003=00
:D004=11
*D D000 D017
:D000=01 00 20 00 11 E8 03 ED /..  ..X.▞
:D008=78 CB DF ED 79 03 1B 7A /xヒ"▞y..z
:D010=B3 20 F4 C9 00 00 00 00 /ウ ホノ....
*■
```

図10.4　00H で修正

すると，プログラムの意味は次のようになります。

マシンコード	ニーモニック
01 00 20	LD BC, 2000H
00	NOP
11 E8 03	LD DE, 1000
ED 78	LOOP : IN A, (C)
CB DF	SET 3, A
ED 79	OUT (C), A
03	INC BC
1B	DEC DE
7A	LD A, D
B3	OR E
20 F4	JR NZ, LOOP
C9	RET

図10.5 修正後のプログラムの意味

4バイト目の**NOP**命令は「何もしない」ので，めでたく望み通りの動作をするプログラムになったわけですね。

10.3 空エリア確保とタイマーへの利用

他の**NOP**命令の使い方としては，マシン語プログラムの設計の途中で，将来書き入れるかもしれない命令のために，場所を確保しておくのにも用いられます。マシン語プログラムでは，一度組んでしまうと格納番地変更が困難になるために，**NOP**で余分の場所を空けておく方法がよく使われます(図10.6)。

また，**NOP**命令がZ80 A（4 MHz）で1 μsという微小実行時間を要することを利用して，周辺LSIとμs単位のタイミングをとって入出力する必要が

アドレス	マシンコード	ニーモニック
D000	11 35 00	LD DE, 53
D003	3E 1D	LD A, 29
D005	DD 21 20 D0	LD IX, 0D020H
D009	00	NOP
D00A	00	NOP
D00B	00	NOP
D00C	DD 74 00	LD (IX+0), H
D00F	DD 75 01	LD (IX+1), L
D012	C9	RET

(D009～D00B の NOP：将来，CALL 命令を書く予定．)

図10.6 メモリー内に「空」をつくるための NOP

あるときの，「時間つぶし」としても用いられます。たとえば，次のサブルーチンは，X1 の HuBASIC の **IOCS**（**入出力制御システム**）中で，カセット・テープの読み書きの際，タイミングをとるために用いられます（このような時間調整のルーチンを**タイマー・ルーチン**といいます）。

アドレス	マシンコード	ニーモニック
0DBF	3E 2E	LD A, 2EH
0DC1	00	NOP
0DC2	3D	LOOP : DEC A
0DC3	C2 C2 0D	JP NZ,LOOP
0DC6	C9	RET

図10.7 タイマー・ルーチンの例

（X1 の HuBASIC IOCS より）

この例で，ループしている部分はAレジスタをカウンタとする「空ループ」であって，BASICで時間かせぎのときによくやる**FOR J＝1 TO 1000：NEXT**のような「空ループ」と同じ原理です。**NOP**命令はループに飛び込む前の1 μsの微小時間つぶしとして機能しています。タイミングが大切なときは，**NOP** 1つであっても重要なのですね！

10.4 HALT命令

halt（**ホールト**と読む）は，「停止する」を意味する英語です。ニーモニックは英単語のまま**HALT**が使われていて，「CPUを停止させる」命令です。マシンコードは76Hです。

BASICでは，画面の最下行（y座標24）に**PRINT**文で何かを表示した後，そのまま終了すると改行が起こって，画面全体が上へ1つズレル（スクロール）現象が起きます。ゲーム画面の試作をしているとき，これでは画面が乱れるので，次のようなテクニックを使うことがあります。

```
10 REM ダイナミック・ストップ
20 '
30 CLS
40 FOR Y=0 TO 24
50   LOCATE 0,Y : PRINT Y;
60 NEXT
70 GOTO 70
```

図10.8 BASICでのダイナミック・ストップ

（turboユーザーの方は**KEYLIST 0：CONSOLE 0,25**を行ってから**RUN**してください。）

70行の**GOTO 70**がミソで，70行を無限にループすることでプログラムが止まりますね。このような止め方を**ダイナミック・ストップ**とよぶことがあります。

さて，HALT 命令というのは，それが書かれている番地でダイナミック・ストップさせる命令です。すなわち，HALT を実行すると CPU はプログラム・カウンタ PC の更新をしなくなりますから，プログラムはそこで停止します。

ここで,「こんな便利な命令があるなら，出し惜しみしないで早く教えてくれればいいのに。プログラム停止のオマジナイ C9H は気持ち悪かった。」と思われる読者がおられるかもしれません。確かに一理あるのですが，X1 シリーズにおいては HALT による止め方では困ることが起こるのです。

HALT による停止中，CPU は 2 つの信号（リセット信号と割り込み信号）を受けつけます。リセット信号は，回路図を見るとハードウェア・システムによる CPU 初期化のために用いられているようで，残念ながら私たちユーザーの出る幕はなさそうです。

もう 1 つの割り込み信号については，私たちが参加できます。割り込みについては次節で述べますが，$\overline{\mathrm{INT}}$ と $\overline{\mathrm{NMI}}$ という 2 つの端子が Z80 の入力端子として設けられています。これらのいずれかの端子を〝0〟レベルにすると，Z80 に割り込みがかかります（割り込みについては，10.6 参照)。

従来の X1 シリーズでは背面に RESET ボタンがあり，turbo シリーズでは前面に NMI というボタンがありますが，これらのボタンを押すと $\overline{\mathrm{NMI}}$ 端子が〝0〟レベルになり，CPU に割り込みがかけられます。HALT 実行中であっても，この割り込みは受けつけられ，システム初期化ルーチンへジャンプして BASIC がホット・スタートします。

では，恐る恐る実験してみましょう。次のプログラムでは最初に HALT を実行しています。プログラム停止を確認するため，続けて「おなじみの」文字 A を表示するプログラムを置いてみます。

アドレス	マシンコード	ニーモニック
D010	76	HALT
D011	01 F4 31	LD BC, 31F4H
D014	3E 41	LD A, 41H
D016	ED 79	OUT (C), A
D018	CB A0	RES 4, B
D01A	3E 07	LD A, 07H
D01C	ED 79	OUT (C), A
D01E	C9	RET

図10.9 HALT 命令の実験

さて，D010H 番地の HALT 命令を実行させます。はたしてプログラムはここで停止するか？

```
MON
*M D010
:D010=76
:D011=01 F4 31
:D014=3E 41
:D016=ED 79
:D018=CB A0
:D01A=3E 07
:D01C=ED 79
:D01E=C9
:D01F=00
*G D010
*█                    A
```

図10.10 実行と確認

アッ！ 意外な結末ですね。HALT ではプログラムは停止せず，次の文字 A 表示プログラムに飛び込んでしまいましたね。HALT 命令についての私たちの理解が正しくないのでしょうか？

10.5　HALT で止まらぬ謎の解明

この謎がはっきり解明できたのは，私にとって本書を執筆している最中でありました。

先の割り込み信号の記述をご覧ください。Z80 CPU には割り込みを受けつける入力端子として $\overline{\mathrm{NMI}}$ と $\overline{\mathrm{INT}}$ の 2 つがあると述べました。$\overline{\mathrm{NMI}}$ の方はすでに説明しましたね。ここで問題になるのは $\overline{\mathrm{INT}}$ 端子です。

X1 シリーズでは turbo も含めて，キーボード関係の入力処理を 2 個のサブ CPU（Z80 ではない！）が担当しています。これらのサブ CPU は，キーボードから何かのキーが押されると，メイン CPU である Z80 の $\overline{\mathrm{INT}}$ 端子を〝0〞レベルにして割り込みをかけます。ところで，モニターの **G** コマンドを用いて **＊ G D010** とした最後に，私たちはリターンキーを押しますね。これにより Z80 に割り込みがかかってしまうのです。ですから，**HALT** を実行する瞬間に解除され，無念！「プログラムは停止しない」というわけであります。

何としてでも **HALT** を実行させたいときは，どうすればよいでしょう。割り込みの詳細は次節で説明しますが，結論を先取りしましょう。

Z80 には「割り込みを禁止する命令」があります。**DI** とニーモニック表記される命令で，マシンコード F3H の 1 バイト命令です。これを **HALT** の前に実行しておくと，キーボード関係からの割り込みを受けつけなくなり，理屈の上から **HALT** が実現するはずです。

さっそく実験です。前のプログラムの直前 D00FH 番地に F3H を書き加えてください。そして，**＊ G D00F** で実行してみましょう。

```
MON
*M D00F
:D00F=F3
:D010=76
*D D00F D01E
:D00F=F3 76 01 F4 31 3E 41 ED /ホv・ホ1>A▞
:D017=79 CB A0 3E 07 ED 79 C9 /yt >・▞y」
*G D00F
```

図10.11　HALT 命令の実行

メデタシ，メデタシ！ HALT が実行されてプログラムは停止しましたね。私たちの予想は正しかったのです。今度は，HALT を解除するのに $\overline{NMI}$ を〝0″レベルにするしかありません（この割り込みは禁止されない！）。背面の RESET ボタン(X1)か，前面の NMI ボタン(turbo)を押して，BASIC のホットスタートへ戻ってください（OK が表示され，カーソルが点滅します）。

ここで，わかったことをまとめておきます。**X1シリーズ（turbo を含め）において，HALT によりプログラム停止をするには，その前に DI 命令を実行する必要がある！**

マシンコード	ニーモニック	
F3	DI	（割り込み禁止）
76	HALT	（CPU停止）

図10.12 X1 シリーズでの HALT の使い方

10.6 割り込みの概念

CPU は，メモリー内に格納されたプログラムを，番地の若い方から順に実行していくのが普通の状態ですが，ジャンプ命令やコール命令，リターン命令などに出会うとプログラムの流れを変えて実行します。これはあくまでプログラムという**ソフトウェア**による流れの変え方ですね。

本節で扱う「割り込み」とは，入出力装置や周辺 LSI などの**ハードウェア**によるプログラム実行の流れの変え方です。

最近の電話に「割り込み電話」というのがありますね。電話会社にその契約をしておくと，通話中に第3者から電話がかかってきたとき，第3者との通話に切り替え，用件をすませて，再びもとの通話に戻るという機能です。この例はコンピュータにおける「割り込み」の概念をかなりはっきり説明していると思います。

たとえば，キー入力を考えましょう。キーボードからの入力はプログラム実行中であっても無視してほしくありません。かといって，プログラム中で「何のキーが押されたか？」をいつも監視するのでは，実行能率が落ちてしまいます。このようなとき，キー入力を「割り込み」により処理するとうまくいきます。すなわち，キー入力がなされた時点で，キーボード（あるいは接続された周辺LSI）がCPUに「割り込み要求」を出します。CPUは要求を受け入れると，実行中のプログラムを中断して，「割り込み処理ルーチン」へジャンプします。処理が終わったら，再びもとのプログラムに戻って続きを実行するという仕組みです。こうすれば，キー入力をプログラム中で監視しなくてもよく能率的です。実際，X1シリーズにおいて，キー入力はサブCPUを経由した割り込みにより処理されます。

10.7 Z80における割り込み

Z80 CPUでは，大別して2種類の**割り込み（interrupt＝インタラプト）**が使われます。1つは**ノンマスカブル・インタラプト**（non-maskable interrupt 略して**NMI**。**マスク不能割り込み**と訳す），もう1つは（**マスカブル・**）**インタラプト**（maskable interrupt。ただ**インタラプト**ということが多く，略して**INT**と書く）とよばれます。

NMIは，プログラム中で絶対に禁止できない最優先の割り込みです。Z80 CPUの$\overline{\text{NMI}}$端子を〝0″レベルにすると，この割り込みがかかります。

これに対して**INT**は，プログラム中でプログラマが割り込みを禁止したり（これを**割り込みをマスクする**という），割り込みのかけ方を設定したりできるものです。Z80の$\overline{\text{INT}}$端子を〝0″レベルにすると，この割り込みがかかります。

したがって，Z80に割り込みをかけようとする周辺LSIや外部スイッチなどは，Z80の$\overline{\text{NMI}}$や$\overline{\text{INT}}$端子に接続しておく必要があります（回路図を詳細に見るとわかります）。

10.8 ノンマスカブル・インタラプト

Z80は$\overline{\mathrm{NMI}}$端子が〝0〞レベルに下がると，現在のプログラム・カウンタ(PC)の内容をスタックに**PUSH**し，新たにPCへ0066Hを書き込みます。すなわち，0066H番地にジャンプします。ということは，**NMI**とは(形の上で)**ハードウェアにより起動されるCALL 0066H命令**であるといえます。

さて，0066H番地からは何らかの「割り込み処理ルーチン」を書いておく必要があります。処理の終わりには，ふつうの**RET**命令ではなく，**RETN命令**を書いておきます。**RETN**はreturn from non-maskable interruptを意味するニーモニックで，マシンコード**ED 45**をもつ2バイト命令です。この命令が実行されると，CPUはスタックに積んでおいた「戻り番地」をPCに**POP**し，割り込みがかかる前に実行中であったプログラムの続きに戻ることになります。

なぜ特別に**RETN**を使うかというと，**NMI**実行中は他のインタラプトは無視されているので，**RETN**によって「もう**NMI**は終了したよ」と周辺に知らせる必要があるからです。

ただし，X1のHuBASICでは**NMI**をシステム初期化(BASICホット・スタート)に使っているので，**RETN**命令は使っていません。実際，0066H番地からダンプすると，次のようにシステム初期化ルーチンへのジャンプ命令が書かれています。

```
MON
*D 0066
:0066=C3 FA 00 63 0E 63 0E 8B /ﾃｻﾞ.c.c.▌
:006E=07 1E 07 63 0E F1 07 A8 /...c.±.ｨ
:0076=07 F7 07 14 08 A1 08 1B /.8......
:007E=07 BF 06 E4 06 1E 07 DE /.ｿ.◆...ﾞ
:0086=05 10 06 63 0E 63 0E E4 /...c.c.◆
:008E=08 63 0E 34 09 63 0E 63 /.c.4.c.c
:0096=0E 81 07 63 0E 3D 09 4A /.▁.c.=.J
:009E=09 63 0E 40 07 64 07 5F /.c.@.d._
:00A6=07 3A 07 00 00 00 00 00 /.:......
:00AE=00 00 00 00 00 00 00 00 /........
:00B6=00 00 00 00 00 00 00 00 /........
:00BE=00 00 00 00 00 00 00 00 /........
:00C6=00 00 00 00 00 00 00 00 /........
:00CE=00 00 00 00 00 00 00 00 /........
:00D6=00 00 00 00 00 00 00 37 /.......7
:00DE=28 2D 34 1F 02 19 1C 00 /(-4.....
*■
```

```
:0066=31 00 00 2A 36 00 E9 00 40 01 40 03 40 FF FF CA /1..*6.▘.@.@.@〒〒ハ
:0076=88 CB 88 FB 8B FD 8B FD 8B 00 EF FE EE E5 F6 FF /|ヒ|年|人|人|.□生▗▞月〒
:0086=FF 02 00 3F 20 00 01 99 10 DF 01 20 12 DF 01 97 /〒..? ..ᒻ.°. .°.┐
:0096=77 DF CD 30 2A 01 74 04 AF CD 53 02 CD 7B 35 ED /w°ヘ0*.t.ッヘS.ヘ{5▞
:00A6=7B 7F 00 CD 82 04 AF 32 38 5C 01 70 17 DF 11 33 /{π.ヘ▂.ッ28¥.p.°.3
:00B6=00 CD E0 04 3A 4F 5C B7 3E 2E 20 02 3E 20 01 91 /.ヘ●.:O¥キ>. .> .|
:00C6=17 DF 01 70 17 DF 21 FF FF 22 85 00 11 00 FF 01 /.°.p.°!〒〒"■...〒.
:00D6=E4 1D DF 30 1D FE 04 20 10 AF 32 F9 FB CD BC 04 /◆.°0.生. .ッ2分年ヘシ.
:00E6=CD 7B 35 CD 69 05 CD 8D 05 AF 01 B6 7D CD 27 02 /ヘ{5ヘi.ヘ■.ッ.カ}ヘ'.
:00F6=18 D4 CD FD 00 18 CF CD 55 00 B7 C8 1B CD 4E 00 /.ヤヘ人..マヘU.キネ.ヘN.
:0106=D2 C7 24 E1 21 E3 F6 E5 CD ED 25 23 22 83 00 36 /メヌ$○!♥月▞ヘ▞%#"▂.6
:0116=00 23 36 00 E1 18 42 E1 2A 85 00 7C A5 3C CA B0 /.#6.○.B○*■.|・<ハー
:0126=00 AF 32 4F 5C 3A 41 5C FE 02 C2 B0 00 C3 5E 08 /.ッ2O¥:A¥生.ツー.テ^.
:0136=CD 82 04 D5 2A 16 00 22 83 00 2A 83 00 CD D2 04 /ヘ▂.ユ*.."▂.*▂.ヘメ.
:0146=28 D5 ED 53 83 00 D1 5E 23 56 23 ED 53 85 00 01 /(ユ▞S▂.ム^#V#▞S■..
:0156=98 6A 3A 38 5C B7 C4 27 02 ED 73 81 00 11 5F 01 /┘j:8¥キト'.▞s▁.._.
```

図10.14 turbo でのノンマスカブル・インタラプト

従来の X1 では，本体背面の **RESET** ボタンを押すことで **NMI** がかけられます。また，turbo では，前面に文字通り **NMI** ボタンが設けられています。これらのボタンを押せば，0066H 番地へのジャンプが行われ，実行中のマシン語プログラムを強制的に終了させられます。

ただし，マシン語プログラムが暴走して 0066H 番地のあたりや，システム初期化ルーチンが壊れてしまうと，この方法は駄目です。X1 の場合は電源を切るか，turbo の場合は IPL ボタンを押すかしなければ，暴走したプログラムは止められません。

10.9 インタラプト

（マスク可能な）インタラプトに関連する命令には，次の 6 つがあります。

IM 0
IM 1 割り込みモード設定（後述）
IM 2

EI … 割り込み許可

DI　　…割り込み禁止（マスク）

RETI …割り込み処理からのリターン

割り込みの**モード**というのは，割り込みのかけ方の指定のことで，Z80では モード0，1，2の3種類があります。

モード0のインタラプトは，Z80の先祖である8080Aが持っていた割り込みのかけ方で，割り込みをかける周辺LSIがCPUに対して，**RST**命令（1バイト）あるいは**CALL**命令（3バイト）を与えることで割り込み処理を実行させる方法です。Z80 CPUは動作を開始した一番最初の状態では，モード0のインタラプトになっています。また，プログラム中で**IM 0**命令（マシンコード**ED 46**）を実行しても，このモードになります。

モード1のインタラプトは，プログラム中で**IM 1**命令（マシンコード**ED 56**）を実行することで指定されます。モード0と異なって，周辺LSI側は割り込み要求（$\overline{\text{INT}}$端子を〝0〟にする）を出すだけでよく，これを受けるとCPU側は自動的に**CALL 0038H**に相当する動作をします。0038H番地からは割り込み処理を書いておきます。この割り込みは，ノンマスカブル・インタラプト（0066H番地へジャンプ）と似ています。

モード2のインタラプトは，動作が複雑なので節を改めて説明します。

これらのインタラプトは，いずれも**禁止（マスク）**することができます。そのための命令が**DI（ディスエーブル・インタラプト＝Disable Interrupt）**というニーモニックで表わされる命令で，マシンコード**F3H**をもつものです。前に**HALT**の項に登場した命令ですね。**DI**命令が実行されると，以後，割り込み許可命令が実行されるまで，Z80は$\overline{\text{INT}}$端子が〝0〟に立ち下がっても割り込み要求を無視します。CPUが動作開始をした最初の状態および割り込み処理中は，**ディスエーブル**の状態になっています。

割り込みを許可する命令が**EI（イネーブル・インタラプト＝Enable Interrupt）**で，マシンコード**FBH**をもちます。Z80の最初の状態は，割り込みモード0かつ割り込み禁止になっているので，モード設定とともに**EI**命令を実行して，割り込みがかかるようにしておく必要があります。また，割り込み

処理中も自動的に割り込み禁止となるので，処理ルーチンから戻る前に **EI** 命令を実行しておかないと，次の割り込みがかからなくなります。

割り込み処理ルーチンからのリターンのためには特別な命令 **RETI (RETurn from Interrupt)** を用います。マシンコード **ED 4D** の2バイト命令です。この命令は，1つの割り込み処理が終了したことを周辺LSIに知らせるためのリターン命令です。

10.10 モード2のインタラプト

X1シリーズ（turboも含めて）で用いられるインタラプトはモード2です。**IM 2** 命令（マシンコード **ED 5E**）を実行すると，このモードになります。この割り込みのかけ方はZ80 CPUの特色の1つで，Z80が制御機器などによく使われる要因になっています。

モード2の割り込みでは，周辺LSI側は割り込み要求を出し，CPUが受けつけると，**ベクトル**とよばれる1バイトのデータをCPU側に出します。CPUはベクトルを受けとり，これを下位8ビットとし，あらかじめ **Iレジスタ（割り込みページ・アドレス・レジスタ）** に書き込まれている1バイトを上位8ビットとして，メモリーのアドレスを指定します。

次に，CPUは上で指定された番地とそれに続く番地の2バイトをジャンプ・アドレスとして割り込み処理ルーチンへジャンプします。

実際に例で見ます。まず，従来のX1のHuBASICにおけるモード2のインタラプトから。ここでは周辺LSIとしてサブCPUが想定され，キー入力割り込みに対応しています。次の番地で，モード2インタラプトの指定がされます。

アドレス	マシンコード	ニーモニック
011B	ED 5E	IM 2

図10.15 X1 HuBASICの割り込みモード設定

また，012DH番地〜013BH番地の処理で，Iレジスタ＝00H，サブCPU側のベクトル＝52Hという設定が行われます。

割り込み準備が完了し，EI命令が実行されました。キーボードのキーを押すと，サブCPUは割り込みをかけます。するとZ80は，Iレジスタの内容(00H)を上位，ベクトル（52H）を下位として構成されるアドレス0052Hを参照します。さて，0052H番地のあたりには，次のようにデータが置かれています。

```
MON
*D 0052 0053
:0052=46 03 D3 03 D3 03 D3 03 /F.E.E.E.
*
```

図10.16 X1 HuBASICの割り込みジャンプ・テーブル

Z80は0052H番地と，次の0053H番地の2バイト0346H（上下位逆転に注

アドレス	マシンコード	ニーモニック
0346	F5	PUSH AF
0347	C5	PUSH BC
0348	D5	PUSH DE
0349	E5	PUSH HL
⋮	⋮	⋮（中略）
03CF	E1	POP HL
03D0	D1	POP DE
03D1	C1	POP BC
03D2	F1	POP AF
03D3	FB	EI
03D4	ED 4D	RETI

図10.17 X1 HuBASICのキー入力割り込み処理

意！）をジャンプ先として，**CALL 0346H** に相当する動作をします。

0346H番地～03D5H番地には，キー入力割り込み処理ルーチンが書かれています。その最初と最後を次に掲げます(図10.17)。

レジスタ類の保護，割り込み許可，**RETI** など割り込み処理に特徴的な姿をしていますね。

10.11　X1 turboの割り込み処理を探す

X1 turboの場合は，ハードウェアも，またturbo BASICもX1よりはるかに複雑で，私自身まだ内部解析が終わっていないのですが，モード2割り込み関連のアドレスを次のように探すことができます。

```
KMODE 0
Ok
MON
*!
)F 0 7FFF ED 5E
:10AC=ED 5E /▖▘^
)D 10AC 10AD
:10AC=ED 5E CD D0 10 CD 3F 6D 21 C7 10 01 00 1C 7E 23 /▖▘^ヘミ.ヘ?m!ヌ....￣#
)D 10D0 10DF
:10D0=21 1A F8 7C ED 47 3E E4 CD 32 14 7D C3 13 14 6F /!.時|▖▘G>♦ヘ2.}テ..o
)D F816 F81F
:F816=2F F8 0F F9 43 F8 72 F8 09 F9 D1 F5 CD 8F 69 EB //時.分C時r時.分ム火ヘ/i▘
)█
```

図10.18　turboの内部探索（割り込み関係）

考え方を以下簡単に説明します。まずturboでは，X1より強力なモニターがIOCS，BIOS ROMとよばれる部分に入っているので，モニターをROMに切り替えます(！コマンド使用)。turboでは，ROMの解析から始めた方がよいようです。

次に，ROMの0000H～7FFFH番地の中で **IM 2**（マシンコード **ED 5E**）を実行している所を探します（Fコマンド使用)。10ACH番地だとわかるので，

このあたりをダンプします。すると，**IM 2**に続き**CALL 10D0H**が実行されていますから，10D0H番地のあたりをダンプすると割り込み関係処理が見つかります(図10.18)。

ROMアドレス	マシンコード	ニーモニック
10D0	**21 1A F8**	**LD HL, 0F81AH**
10D3	**7C**	**LD A, H** ｝Iレジスタを F8Hに設定
10D4	**ED 47**	**LD I, A**
10D6	**3E E4**	**LD A, 0E4H** ｝サブCPUにベクトル1AHを設定
10D8	**CD 32 14**	**CALL 1432H**
10DB	**7D**	**LD A, L**
10DC	**C3 13 14**	**JP 1413H**

図10.19 turboのIレジスタ，ベクトル設定

この部分を解読すると，割り込みジャンプ・テーブルはRAMエリアのF81AH番地の付近にあるとわかるので，このあたりをダンプします（図10.18参照)。2バイトずつジャンプ・テーブルらしきものが並んでいますね。このうち，F81AH番地と，その次の番地にあるF843Hがジャンプ先です。こうして，F843H番地からダンプすると，ありましたね！

```
)D F843 F874
:F843=E5 21 FF F5 F5 C5 01 00 0B ED 78 F5 F6 01 ED 79 /♠!ﾃｽｽｵ...▚x火月.▚y
:F853=3E 1A DB 01 F5 3E 1D D3 00 CD 75 F8 F1 E6 10 3E />.□.ｽ>.ﾓ.^u時土◢.>
:F863=1D 28 01 3C D3 00 F1 01 00 0B ED 79 C1 F1 E1 FB /.(.<ﾓ.ｷ...▚yﾁ土0年
:F873=ED 4D E9 8A 0F 0E 11 10 7F 02 C7 00 50 18 76 76 /▖M▘▌....π.ﾇ.P.vv
)█
```

図10.20 turbo BASICの割り込み処理（サブCPU）

これがサブCPU関係のキー入力割り込み処理ルーチンです。最初の方でレジスタ類のPUSHがされ，終わりの方で**POP**がなされています。F873H番地から2バイト**ED 4D**は，**RETI**のマシンコードですね。

こうした探索を突破口にturboのBASICを解読していくと，0000H番地から順にするより興味が持続できるのではないでしょうか？　あとの作業は，ガッツのあるturboユーザーの皆さんにおまかせします。

第11章　BASICとマシン語の共存

本書では，マシン語が主役であり，BASICプログラムは数えるほどしか登場していません。しかし，よく考えてみると，私たちがマシン語プログラムを格納したり，実行するために使ってきたモニターは，BASICシステムの中に組み込まれていたのであり，目立たなくてもBASICは私たちの「マシン語環境」として存在していたのです。

本章では，私たちユーザーのマシン語プログラムが，BASICとうまく共存していくために最低限知っていなければならないことを説明して，本書の結びとします。

11.1　変な現象

まず，1つの実験をおめにかけましょう。次は，第2章で登場した最も基本的なプログラムです。

```
LD   A, 40H
LD   (0D010H), A
RET
```

では，モニターを起動し，マシン・コードをFF00H番地から格納してみましょう。

格納できたら，確認のためダンプします。

```
*M FF00
:FF00=3E 40
:FF02=32 10 D0
:FF05=C9
:FF06=43
*D FF00
:FF00=2A 44 20 46 46 30 30 00 /*D FF00.
:FF08=00 00 00 00 00 00 00 00 /........
:FF10=00 00 00 00 00 00 00 00 /........
:FF18=00 00 00 00 00 00 00 00 /........
:FF20=00 00 00 00 00 00 00 00 /........
:FF28=00 2A 85 2C 85 80 2F 8A /.*▄,▄_/▌
:FF30=83 32 8B 34 8A 84 37 8A /▄2▌ 4▌▄7▌
:FF38=39 3A 3B 8E 3D 8B 81 82 /9:;█=▌_▁
:FF40=84 88 89 87 89 87 86 8D /▄ ▌ ██ ████
:FF48=8E 8D 8D 80 81 80 80 00 /███____.
:FF50=8F 8F 8F 8F 8F 8F 8F 8F ////////
:FF58=8F 8F 8F 8F 8F 8F 8F 8F ////////
:FF60=8F 8F 8F 8F 8F 8F 8F 8F ////////
:FF68=8F 8F 8F 8F 8F 8F 8F 8F ////////
:FF70=8F 8F 8F 8F 8F 8F 8F 8F ////////
:FF78=8F 8F 8F 8F 8F 8F 8F 8F ////////
*█
```

図11.1 変な現象

いかがですか？ メモリーに書き込んだはずのマシン・コードが全く別のものに「化けて」しまっていますね。これはどうしたことなのでしょう。

本章の冒頭で，BASICが「マシン語環境」として存在していると述べましたが，上の現象はその1つの現れです。

モニターを起動すると，FF00H番地以降は，モニター用の**ワークエリア(作業領域)**として使われます。とくに，最初の方にはキーボードより入力した，モニターコマンドの文字列が格納されます。図11.1でFF00H番地からFF06H番地までに格納されている文字列がまさにそうですね。このために，せっかく入力したマシン・コードが書き変えられてしまったというわけです。

このように，私たちのマシン語プログラムをメモリー上のどこに格納するかは，マシン語プログラムの入力・実行を管理しているシステム（私たちの場合は，BASICです。）と切り離して考えることはできません。

11.2 メモリー・マップとマシン語エリア

私たちが今まで作ってきたマシン語プログラムは，どれでも短いものばかりでした。第10章まで読み進んだ皆さんは，Z80の命令を一通りマスターしたわけですから，これらを組み合わせて，もう少し長いプログラムに挑戦してみようと思っておられることでしょう。本節では，このようなときの注意を述べます。

次の図をご覧ください。

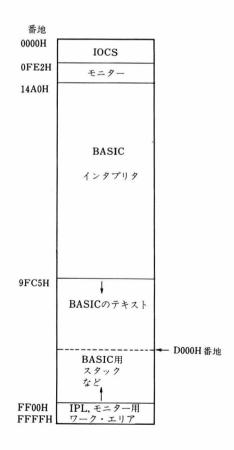



図11.2 X1 HuBASIC (CZ-8CB01) のメモリー・マップ

このような図は**メモリー・マップ**とよばれ，システムがメモリー内のどこを使用しているかを示すものです。図11.2はテープBASIC（CZ-8CB01）のメモリー・マップです。これを見ると，私たちがプログラムを書いてきたD000H番地は，上下からBASIC使用エリアにはさまれた不安定な領域であったことが判明します。

私たちは，本書で長いBASICプログラムを作成しなかったし，またマシン語プログラムも短いものでしたから，BASIC使用エリアと衝突せずに済んでいたといえます。このような偶然に頼らず，はっきりと「マシン語プログラムを格納する領域」を確保するには，どうすればよいのでしょう。

BASICの命令に**CLEAR**というのがあります。この命令には「変数をクリアする」機能のほかに，上のようなBASIC使用エリアを番地の高い方から制限する働きがあります。

CLEAR アドレス値

という命令をダイレクトに，あるいはプログラム中で実行すると，以後，指定されたアドレス値－1よりも，番地の若い方がBASIC使用エリアとなり，指定アドレス値以降は自由に使えるエリアとなります。ここが私たちユーザーのマシン語プログラムを格納できる場所です。

たとえば，D000H番地以降をユーザー・エリアとするには，**CLEAR &HD000**を実行します。

すると，メモリー・マップは次のように変化します。

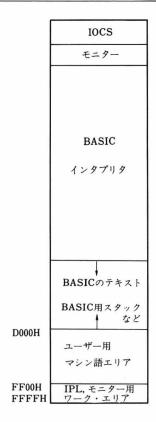



図11.3 CLEAR &HD000 実行後のメモリー・マップ

ですから，私たちが第10章までに行なってきた実験の前には，メモリー・マップをこのような状態にしておくのが「正式」であったわけです。

ただし，D000H 番地以降がすべて自由なわけではなく，FF00H～FFFFH 番地はモニター使用時に使われてしまうので，この領域も除外しなくてはなりません。さらに，turbo BASIC（CZ-8FB02）を使うときには，F800H 番地～FFFFH 番地が「IPL および BIOS ワークエリア」としてとられてしまいます。カナ漢字変換機能を用いるときには，F000H 番地以降もシステムにとられます。

したがって，D000H～EFFFH 番地のあたりに，ユーザー用マシン語エリア

を設定しておくのが安全といえましょう(開始番地 D000H も状況によって，前後にずらさなければならないときもあります)。

BASIC使用エリアを制限する命令には，もう1つ **NEWON** 命令があります。

NEWON **アドレス値**

と指定すると，BASICのテキスト(いわゆる私たちがBASICで組むプログラム）を格納するエリアが指定アドレス値以降に制限されます。たとえば，

NEWON &HB000

を実行すると，

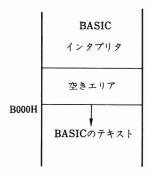



図11.4 NEWON &HB000実行後のメモリー・マップ

この図のように，BASICインタプリタ本体とBASICテキストの間に空エリアが生じるので，ここにユーザーのマシン語プログラムを格納できます。

11.3 BASICからマシン語サブルーチンを呼び出す方法

私たちは，今までマシン語プログラムの実行はすべてモニターから **G** コマンドで行ってきました。しかし，BASICのコマンド•レベルからマシン語プログラムを走らせることもできます。

その1つの方法は，BASICの**CALL**命令を使うものです。マシン語のニーモニックと同じ名称ですが，こちらはあくまでBASICの命令であり，ダイレクトでもプログラム中でも使えます。

CALL [アドレス値]

とすると，指定アドレス値から始まるマシン語サブルーチンを呼び出し，サブルーチン内の**RET**命令で再びBASICに戻ってきます。第10章までに作成したプログラムを，BASICから**CALL**命令で実行してみましょう。

例として，第6章で学んだ「文字Aを画面中央に表示するプログラム」をBASICより実行します。

```
CLEAR &HD000
Ok
MON
*M D000
:D000=01 F4 31
:D003=3E 41
:D005=ED 79
:D007=01 F4 21
:D00A=3E 07
:D00C=ED 79
:D00E=C9
:D00F=00
*R                         A
Ok
CALL &HD000
Ok
█
```

図11.5　BASICの**CALL**命令

もう1つの方法は，BASICの**USR**関数を使います。**CALL**命令と違って，**USR**関数はBASICからマシン語サブルーチンへ，レジスタ経由でデータを渡すことができます。

USR関数は，**USR 0**から**USR 9**まで10個使うことができます。単に**USR**とすると，**USR 0**と同じことになります。

使用にあたっては，**USR**関数が呼び出すマシン語サブルーチンの番地をあらかじめ定義しておく必要があります。ここでは，**USR**関数の1番がD000H

番地からのマシン語サブルーチンを呼び出すものとしましょう。次のようにします。

DEFUSR1＝&HD000

さて，上で定義したUSR1は，次の代入文の形で使われます。

数値変数＝USR1（数式）
　あるいは
文字変数＝USR1（文字式）

（　）内の数式や文字式が，引数としてマシン語サブルーチンへ渡されるデータです。

USR関数が呼び出されると，Aレジスタと HLレジスタペアに次のように値がセットされます。

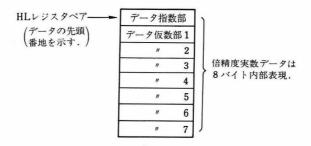
① 引数が整数型（数式）のとき
Aレジスタ＝02H（データのバイト数）

HLレジスタペア——→
（データの先頭番地を示す．）

データ下位バイト
データ上位バイト

整数データは2バイト．

② 引数が単精度実数型（数式）のとき
Aレジスタ＝05H（データのバイト数）

③ 引数が倍精度実数型（数式）のとき
Aレジスタ=08H（データのバイト数）

④ 引数が文字型（文字式）のとき

Aレジスタ=03H

Bレジスタ=文字データの長さ

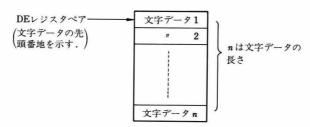



図11.6　USR関数呼び出し時のレジスタ内容

ただし，文字式を引数とした場合のみは特別で，Bレジスタ，DEレジスタペアの方に実際のデータの情報が入れられます。

ですから，呼び出されたマシン語サブルーチン側では，これらのレジスタの情報を活用して処理を行うことができますね。

ここでは簡単な実験として，整数型引数の場合を考えましょう。D000H番地

から格納するマシン語サブルーチンは

アドレス	マシンコード	ニーモニック
D000	34	INC (HL)
D001	C9	RET

というわずか2バイトのプログラムで，引数として渡される整数データの下位バイトを＋1するものです。マシンコードをモニターで入力した後，次のBASICプログラムを実行します。

```
CLEAR &HD000
Ok
MON
*M D000
:D000=34
:D001=C9
:D002=00
*R
Ok
10 DEFUSR1=&HD000
20 INPUT "ｾｲｽｳ = ",A%
30 Y%=USR1(A%)
40 PRINT Y%
50 GOTO 20
RUN
ｾｲｽｳ = 1
 2
ｾｲｽｳ = 255
 0
ｾｲｽｳ = ■
```

図11.7 USR関数の使用例

整数型データであることを確実にするため，上の例では変数名の後に%をつけて整数型変数にしています。引数A%のデータは加工されて(＋1されて)，変数Y%に代入されます。ただし，上位バイトへの繰り上がりを考慮していませんから，255を入力すると0になって返ってきます。

このように，**USR**関数ではBASICからマシン語サブルーチンへデータを渡し，加工されて返ってくるデータを再びBASICで受け取ることができるのです。

あとがき

私が最初のX1（CZ-800C）で，マシン語の勉強を始めてから3年近くになります。初めの頃，X1のマシン語の本は皆無でしたから，私は主としてPC-8001用に書かれたマシン語入門書で勉強してきました。

他機種用のマシン語プログラムの移植は困難であると言われていますし，私もその意見に基本的には賛成です。しかし，X1のマシン語学習の出発点において，私には他機種用の「お手本」しかありませんでした。しかし，それだからこそ，すべての「実験」がスリリングで新鮮でした。私にとっては「発見」の連続といった毎日でした。メモリー内にデータを格納する数バイトのマシン語プログラムが動いたときの感動を，私は今でも忘れません。HALT命令でプログラムが止まらない謎に悩まされた日々，初めて画面表示に成功した日……私は，このようなワクワクするような経験を「日記」としてつけてきました。

その頃のメモを今になって読み返すと，「何故こんなにマシン語に夢中になれたのだろう!?」と不思議になるほど，「熱」が伝わってきます。そうした物思いにふけりながら私は，次のようなことを考えました。

今まで知らなかったこと，あるいはわからなかったことを，「わかっていく」過程というのは，どんなものなのでしょう。この問題はきっと人によって千差万別でしょうが，ここでは私の体験に基づいて考えてみます。

よく最初の状態として，「白紙」あるいは「空」，「無」というものが想定されることがあります。「心を虚しくして」という表現も使われます。けれども，マシン語学習の出発点において，私の頭の中は決して「白紙状態」ではなかったようです。たとえば，Aレジスタといえば「変数A」を思い浮べているというように，私の考えは主としてBASICからくる「偏見」に満ち満ちていました。そのためにたくさんの失敗をしました。BASICではEND文を書かなくても，

多くの場合，プログラムは止まりますが，それをそのままマシン語に直して見事に「暴走」したことも何度となくあります。

しかしながら，こういう試行錯誤を経て，私の中の「偏見」あるいは「思い込み」は少しずつ軌道修正されていきました。後になって考えてみると，そのような「偏見」の数々は，作業仮説としてマシン語プログラム作りの原動力になっていたように思います。

本書の執筆をしながら，私はこのような「マシン語学習過程」を何とか活かせないものだろうか？　と考え続けました。もちろん，試行錯誤のままを再現したら，わけのわからぬものになるのは明らかです。いろいろ考えた末，Z80の命令をグループごとにまとめて解説するという「入門書」スタイルをとりつつも，命令の並べ方と叙述の展開をできるだけ「思考の順序」に合わせるという方法を採用することにしました。普通のZ80の解説書では，入出力命令は最後の方に配置されることが多いようですが，テキスト画面を題材とした入出力命令が「目で結果を確認できる」という重要な特質を持つことに注目し，本書では第6章に配置されているのも，その一例です。また，各章内では「実験→結果の確認」という手続きを執拗に繰り返していますが，それも上述の執筆意図から来るものです。

Z80の解説書・入門書はすでに溢れるほど出版されていますが，本書が少しでも「これからX1シリーズあるいはturboシリーズでマシン語を始めよう」とする方々のお役に立つことを祈って「あとがき」といたします。

最後になりましたが，本書の出版を実現してくださった工学図書株式会社の皆様に感謝の意を表し，筆をおきます。

1985年夏　　清水　保弘

付録1. Ｚ80命令表(機能別)

8 ビットロード命令

＼ ×	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(mn)	n	I	R
LD A, ×	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n m	3E n	ED 57	ED 5F
LD B, ×	47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n		
LD C, ×	4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n		
LD D, ×	57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n		
LD E, ×	5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n		
LD H, ×	67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n		
LD L, ×	6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n		
LD (HL), ×	77	70	71	72	73	74	75							36 n		
LD (BC), ×	02															
LD (DE), ×	12															
LD (IX+d), ×	DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n		
LD (IY+d), ×	FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n		
LD (mn), ×	32 n m															
LD I, ×	ED 47															
LD R, ×	ED 4F															

16ビットロード命令

＼ ×	AF	BC	DE	HL	SP	IX	IY	mn	(mn)
LD AF, ×									
LD BC, ×								01 n m	ED 4B n m
LD DE, ×								11 n m	ED 5B n m
LD HL, ×								21 n m	2A n m
LD SP, ×				F9		DD F9	FD F9	31 n m	ED 7B n m
LD IX, ×								DD 21 n m	DD 2A n m
LD IY, ×								FD 21 n m	FD 2A n m
LD (mn), ×	ED 43 n m	ED 53 n m	22 n m	ED 73 n m	DD 22 n m	FD 22 n m			
PUSH ×	F5	C5	D5	E5		DD E5	FD E5		
POP ×	F1	C1	D1	E1		DD E1	FD E1		

交換・ブロック命令

交　換

EX AF, AF′	08
EX DE, HL	EB
EX (SP), HL	E3
EX (SP), IX	DD E3
EX (SP), IY	FD E3
EXX	D9

ブロック転送

LDI	ED A0
LDIR	ED B0
LDD	ED A8
LDDR	ED B8

ブロックサーチ

CPI	ED A1
CPIR	ED B1
CPD	ED A9
CPDR	ED B9

8ビット算術・論理演算命令

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
ADD A, ×	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADC A, ×	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUB ×	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SBC A, ×	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
AND ×	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
XOR ×	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
OR ×	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
CP ×	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INC ×	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DEC ×	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

16ビット算術演算命令

×	BC	DE	HL	SP	IX	IY
ADD HL, ×	09	19	29	39		
ADD IX, ×	DD 09	DD 19		DD 39	DD 29	
ADD IY, ×	FD 09	FD 19		FD 39		FD 29
ADC HL, ×	ED 4A	ED 5A	ED 6A	ED 7A		
SBC HL, ×	ED 42	ED 52	ED 62	ED 72		
INC ×	03	13	23	33	DD 23	FD 23
DEC ×	0B	1B	2B	3B	DD 2B	FD 2B

ビット操作命令

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
BIT 0, ×	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
BIT 1, ×	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
BIT 2, ×	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
BIT 3, ×	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
BIT 4, ×	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
BIT 5, ×	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
BIT 6, ×	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
BIT 7, ×	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RES 0, ×	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
RES 1, ×	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
RES 2, ×	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
RES 3, ×	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
RES 4, ×	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
RES 5, ×	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
RES 6, ×	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
RES 7, ×	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET 0, ×	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
SET 1, ×	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
SET 2, ×	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
SET 3, ×	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
SET 4, ×	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
SET 5, ×	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
SET 6, ×	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
SET 7, ×	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

アキュムレータ・フラグ・CPU制御命令

アキュムレータ・フラグ制御

DAA	27
CPL	2F
NEG	ED 44
CCF	3F
SCF	37

CPU制御

NOP	00
HALT	76
DI	F3
EI	FB
IM 0	ED 46
IM 1	ED 56
IM 2	ED 5E

ローテイト・シフト命令

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
RLC ×	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
RRC ×	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
RL ×	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
RR ×	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
SLA ×	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
SRA ×	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
SRL ×	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
RLD								ED 6F		
RRD								ED 67		

RLCA	07
RRCA	0F
RLA	17
RRA	1F

ジャンプ命令

×	無条件	キャリー	ノンキャリー	ゼロ	ノンゼロ	パリティ偶数	パリティ奇数	負	正	カウント
		C	NC	Z	NZ	PE	PO	M	P	
JP ×, mn	C3 n m	DA n m	D2 n m	CA n m	C2 n m	EA n m	E2 n m	FA n m	F2 n m	
JR ×, e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JP (HL)	E9									
JP (IX)	DD E9									
JP (IY)	FD E9									
DJNZ e										10 e-2

コール・リターン命令

コール・リターン

×	無条件	キャリー	ノンキャリー	ゼロ	ノンゼロ	パリティ偶数	パリティ奇数	負	正
		C	NC	Z	NZ	PE	PO	M	P
CALL ×, mn	CD n m	DC n m	D4 n m	CC n m	C4 n m	EC n m	E4 n m	FC n m	F4 n m
RET ×	C9	D8	D0	C8	C0	E8	E0	F8	F0
RETI	ED 4D								
RETN	ED 45								

リスタート

×	00H	08H	10H	18H	20H	28H	30H	38H
RST ×	C7	CF	D7	DF	E7	EF	F7	FF

入出力命令

入力

×	A	B	C	D	E	H	L
IN ×, (n)	DB n						
IN ×, (C)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69

OUTI	ED A2	ブロック入力コマンド
OTIR	ED B2	
OUTD	ED AA	
OTDR	ED BA	

出力

×	A	B	C	D	E	H	L
OUT (n), ×	D3 n						
OUT (C), ×	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69

OUTI	ED A3	ブロック出力コマンド
OTIR	ED B3	
OUTD	ED AB	
OTDR	ED BB	

付録2．Z80命令表（アルファベット順）

用いる記号

○ n は　8ビット定数

○mnは　16ビット定数，mが上位8ビット，nが下位8ビット
（ニーモニックではラベル名を書いてもよい）

○ d は　−128〜+127の範囲（符号付1バイト16進数）のディスプレイスメント。

○ e は　−126〜+129の範囲の1バイト相対アドレス。命令が書かれているアドレスを0として数える。
（ニーモニックではラベル名や絶対アドレスを書いてもよい。ただし，アセンブラにより異なる場合もある）

○Cyは　キャリーフラグ

○添字Hは上位8ビット（high），添字Lは下位8ビット（low）を意味する。

○ビット操作命令 SET, RES, BIT で，ビット番号は下記のとおりである。

7	6	5	4	3	2	1	0

左端 ↑第7ビット（ビット7）　右端 ↑第0ビット（ビット0）

○X1ではI/Oポートが64Kバイトあるので，特に次の記号を用いる。
I/O（BC）は，BCレジスタペアの内容番地のポートを表わす。
I/O（An）は，Aレジスタを上位8ビット，定数nを下位8ビットとする番地のポートを表わす。

○フラグ変化

×　不定
1　1になる（セットされる）
0　0になる（リセットされる）
・　状態に従って，セット・リセットされる
−　変化せず

ニーモニック	マシンコード	S	Z	H	P/V: P	P/V: V	N	C	所要クロックサイクル	動作
ADC A, n	CE n								7	8ビット加算キャリー付
ADC A, A	8F									(アド・ウィズ・キャリー)
ADC A, B	88									A←A+(ソース)+Cy
ADC A, C	89									
ADC A, D	8A								4	
ADC A, E	8B	·	·	·	—	·	0	·		
ADC A, H	8C									
ADC A, L	8D									
ADC A, (HL)	8E								7	
ADC A, (IX+d)	DD 8E d								19	
ADC A, (IY+d)	FD 8E d								19	
ADC HL, BC	ED 4A									16ビット加算キャリー付
ADC HL, DE	ED 5A	·	·	×	—	·	0	·	15	(アド・ウィズ・キャリー)
ADC HL, HL	ED 6A									HL←HL+(ソース)+Cy
ADC HL, SP	ED 7A									
ADD A, n	C6 n								7	8ビット加算(アド)
ADD A, A	87									
ADD A, B	80									
ADD A, C	81									
ADD A, D	82								4	A←A+(ソース)
ADD A, E	83	·	·	·	—	·	0	·		
ADD A, H	84									
ADD A, L	85									
ADD A, (HL)	86								7	
ADD A, (IX+d)	DD 86 d								19	
ADD A, (IY+d)	FD 86 d								19	
ADD HL, BC	09									16ビット加算(アド)
ADD HL, DE	19								11	HL←HL+(ソース)
ADD HL, HL	29									
ADD HL, SP	39									
ADD IX, BC	DD 09									
ADD IX, DE	DD 19	—	—	×	—	—	0	·	15	IX←IX+(ソース)
ADD IX, IX	DD 29									
ADD IX, SP	DD 39									

ニーモニック	マシンコード	S	Z	H	P/V (P)	P/V (V)	N	C	所要クロックサイクル	動作
ADD IY, BC	FD 09	−	−	×	−	−	0	·	15	IY←IY+(ソース)
ADD IY, DE	FD 19									
ADD IY, IY	FD 29									
ADD IY, SP	FD 39									
AND n	E6 n	·	·	1	·	−	0	0	7	論理積(アンド)
AND A	A7								4	A←A∧(ソース)
AND B	A0									
AND C	A1									
AND D	A2									
AND E	A3									
AND H	A4									
AND L	A5									
AND (HL)	A6								7	
AND (IX+d)	DD A6 d								19	
AND (IY+d)	FD A6 d								19	
BIT 0, A	CB 47	×	·	1	×	−	0	−	8	ビットテスト ソースの第0ビットを調べZフラグを設定
BIT 0, B	CB 40									
BIT 0, C	CB 41									
BIT 0, D	CB 42									
BIT 0, E	CB 43									
BIT 0, H	CB 44									
BIT 0, L	CB 45									
BIT 0, (HL)	CB 46								12	
BIT 0, (IX+d)	DD CB d 46								20	
BIT 0, (IY+d)	FD CB d 46								20	
BIT 1, A	CB 4F	×	·	1	×	−	0	−	8	ビットテスト ソースの第1ビットを調べZフラグを設定
BIT 1, B	CB 48									
BIT 1, C	CB 49									
BIT 1, D	CB 4A									
BIT 1, E	CB 4B									
BIT 1, H	CB 4C									
BIT 1, L	CB 4D									
BIT 1, (HL)	CB 4E								12	
BIT 1, (IX+d)	DD CB d 4E								20	
BIT 1, (IY+d)	FD CB d 4E								20	

a	b	a∧b
0	0	0
0	1	0
1	0	0
1	1	1

ニーモニック	マシンコード	フラグ変化 S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動作
BIT 2, A	CB 57									ビットテスト
BIT 2, B	CB 50									ソースの第2ビットを調べZフラグを設定
BIT 2, C	CB 51									
BIT 2, D	CB 52								8	
BIT 2, E	CB 53									
BIT 2, H	CB 54	×	·	1	×	—	0	—		
BIT 2, L	CB 55									
BIT 2, (HL)	CB 56								12	
BIT 2, (IX+d)	DD CB d 56								20	
BIT 2, (IY+d)	FD CB d 56								20	
BIT 3, A	CB 5F									ビットテスト
BIT 3, B	CB 58									ソースの第3ビットを調べZフラグを設定
BIT 3, C	CB 59									
BIT 3, D	CB 5A								8	
BIT 3, E	CB 5B									
BIT 3, H	CB 5C	×	·	1	×	—	0	—		
BIT 3, L	CB 5D									
BIT 3, (HL)	CB 5E								12	
BIT 3, (IX+d)	DD CB d 5E								20	
BIT 3, (IY+d)	FD CB d 5E								20	
BIT 4, A	CB 67									ビットテスト
BIT 4, B	CB 60									ソースの第4ビットを調べZフラグを設定
BIT 4, C	CB 61									
BIT 4, D	CB 62								8	
BIT 4, E	CB 63									
BIT 4, H	CB 64	×	·	1	×	—	0	—		
BIT 4, L	CB 65									
BIT 4, (HL)	CB 66								12	
BIT 4, (IX+d)	DD CB d 66								20	
BIT 4, (IY+d)	FD CB d 66								20	
BIT 5, A	CB 6F									ビットテスト
BIT 5, B	CB 68									ソースの第5ビットを調べZフラグを設定
BIT 5, C	CB 69	×	·	1	×	—	0	—	8	
BIT 5, D	CB 6A									
BIT 5, E	CB 6B									

ニーモニック	マシンコード	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
BIT 5, H	CB 6C									
BIT 5, L	CB 6D									
BIT 5, (HL)	CB 6E								12	
BIT 5, (IX+d)	DD CB d 6E								20	
BIT 5, (IY+d)	FD CB d 6E								20	
BIT 6, A	CB 77									ビットテスト ソースの第6ビットを調べZフラグを設定
BIT 6, B	CB 70									
BIT 6, C	CB 71									
BIT 6, D	CB 72								8	
BIT 6, E	CB 73									
BIT 6, H	CB 74	×	·	1	×	−	0	−		
BIT 6, L	CB 75									
BIT 6, (HL)	CB 76								12	
BIT 6, (IX+d)	DD CB d 76								20	
BIT 6, (IY+d)	FD CB d 76								20	
BIT 7, A	CB 7F									ビットテスト ソースの第7ビットを調べZフラグを設定
BIT 7, B	CB 78									
BIT 7, C	CB 79									
BIT 7, D	CB 7A								8	
BIT 7, E	CB 7B									
BIT 7, H	CB 7C	×	·	1	×	−	0	−		
BIT 7, L	CB 7D									
BIT 7, (HL)	CB 7E								12	
BIT 7, (IX+d)	DD CB d 7E								20	
BIT 7, (IY+d)	FD CB d 7E								20	
CALL NZ, mn	C4 n m									サブルーチン・コール(条件付) ・条件が成立すれば戻り番地〔PC〕をスタックへPUSHしmnへジャンプ〔PC←mn〕 ・成立しなければ本命令は無視する
CALL Z, mn	CC n m									
CALL NC, mn	D4 n m								成立時	
CALL C, mn	DC n m								17	
CALL PO, mn	E4 n m	−	−	−	−	−	−	−	不成立	
CALL PE, mn	EC n m								10	
CALL P, mn	F4 n m									
CALL M, mn	FC n m									
CALL mn	CD n m	−	−	−	−	−	−	−	17	サブルーチン・コール(無条件) PCをスタックへ PUSH し PC←mn

ニーモニック	マシンコード	S	Z	H	P/V		N	C	所要クロックサイクル	動作
					P	V				
CCF	3F	−	−	×	−	−	0	・	4	Cyを反転〔Cy←$\overline{Cy}$〕
CP n	FE n								7	比較(コンペア) A−(ソース)の演算をするAの内容は変わらずフラグだけが変化する
CP A	BF									
CP B	B8									
CP C	B9									
CP D	BA								4	
CP E	BB	・	・	・	−	・	1	・		
CP H	BC									
CP L	BD									
CP (HL)	BE								7	
CP (IX+d)	DD BE d								19	
CP (IY+d)	FD BE d								19	
CPD	ED A9	×	・	×	−	・	1	−	16	比較(コンペア・デクリメント) A−(HL)のフラグ変化のみ　HL←HL−1, BC←BC−1
CPDR	ED B9	×	・	×	−	・	1	−	1バイトにつき 21 最終のみ 16	比較(コンペア・デクリメント・リピート) CPDをA=(HL)〔Zフラグ=1〕またはBC=0〔Vフラグ=0〕までくり返す
CPI	ED A1	×	・	×	−	・	1	−	16	比較(コンペア・インクリメント) A−(HL)のフラグ変化のみ　HL←HL+1, BC←BC−1
CPIR	ED B1	×	・	×	−	・	1	−	1バイトにつき 21 最終のみ 16	比較(コンペア・インクリメント・リピート) CPIをA=(HL)〔Zフラグ=1〕またはBC=0〔Vフラグ=0〕までくり返す
CPL	2F	−	−	1	−	−	1	−	4	コンプリメント Aレジスタに対しビット反転　0→1　1→0
DAA	27	・	・	・	・	−	−	・	4	デシマル・アジャスト・アキュムレータ Aレジスタに対し10進補正

ニーモニック	マシンコード	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V		N	C		
					P	V				
DEC A	3D	·	·	·	—	·	1	—	4	8ビット デクリメント (ソース)←(ソース)−1
DEC B	05									
DEC C	0D									
DEC D	15									
DEC E	1D									
DEC H	25									
DEC L	2D									
DEC (HL)	35								11	
DEC (IX+d)	DD 35 d								23	
DEC (IY+d)	FD 35 d								23	
DEC BC	0B	—	—	—	—	—	—	—	6	16ビット デクリメント (ソース)←(ソース)−1
DEC DE	1B									
DEC HL	2B									
DEC SP	3B									
DEC IX	DD 2B								10	
DEC IY	FD 2B								10	
DI	F3	—	—	—	—	—	—	—	4	割り込み禁止(ディスエーブル・インタラプト)
DJNZ e	10 e−2	—	—	—	—	—	—	—	B＝0 8 B≠0 13	(デクリメント・ジャンプノンゼロ) B←B−1　B≠0ならeバイトだけジャンプ〔PC←PC+e〕 B＝0ならジャンプせず〔e＝0と同じ〕
EI	FB	—	—	—	—	—	—	—	4	割り込み許可(インネーブル・インタラプト)
EX (SP), HL	E3	—	—	—	—	—	—	—	19	交換(エクスチェンジ) ソースとデスティネーションの内容を交換する
EX (SP), IX	DD E3								23	
EX (SP), IY	FD E3								23	
EX AF, AF′	08								4	
EX DE, HL	EB								4	
EXX	D9	—	—	—	—	—	—	—	4	BC DE HLの内容とBC’DE’HL’の内容を交換する
HALT	76	—	—	—	—	—	—	—	4	命令実行の進行を止めリセットまたは割り込み待ちとなる(ホルト)

ニーモニック	マシンコード	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V		N	C		
					P	V				
IM 0	ED 46	—	—	—	—	—	—	—	8	割り込みモードを0に設定する
IM 1	ED 56								8	割り込みモードを1に設定する
IM 2	ED 5E								8	割り込みモードを2に設定する
INC A	3C	・	・	・	—	・	0	—	4	8ビットインクリメント (ソース)←(ソース)+1
INC B	04									
INC C	0C									
INC D	14									
INC E	1C									
INC H	24									
INC L	2C									
INC (HL)	34								11	
INC (IX+d)	DD 34 d								23	
INC (IY+d)	FD 34 d								23	
INC BC	03	—	—	—	—	—	—	—	6	16ビットインクリメント (ソース)←(ソース)+1
INC DE	13									
INC HL	23									
INC SP	33									
INC IX	DD 23								10	
INC IY	FD 23								10	
IN A, (C)	ED 78	・	・	0	・	—	0	—	12	入力 BCレジスタの内容番地のポートからデスティネーションのレジスタへ入力 〔デスティネーション←I/O(BC)〕
IN B, (C)	ED 40									
IN C, (C)	ED 48									
IN D, (C)	ED 50									
IN E, (C)	ED 58									
IN H, (C)	ED 60									
IN L, (C)	ED 68									
IN A, (n)	DB n	—	—	—	—	—	—	—	11	入力 A←I/O(An)
IND	ED AA	×	・	×	×	—	×	×	16	イン・デクリメント (HL)←I/O(BC), B←B−1 HL←HL−1

ニーモニック	マシンコード	フラグ変化 S	Z	H	P/V P	V	N	C	所要クロックサイクル	動作
INDR	ED BA	×	1	×	×	−	×	×	1バイトにつき 21 最終のみ 16	イン・デクリメント・リピート INDをB=0までくり返す
INI	ED A2	×	·	×	×	−	×	×	16	イン・インクリメント (HL)←I/O(BC), HL←HL+1 B←B−1
INIR	ED B2	×	1	×	×	−	×	×	1バイトにつき 21 最終のみ 16	イン・インクリメント・リピート INIをB=0までくり返す
JP (HL) JP (IX) JP (IY) JP mn	E9 DD E9 FD E9 C3 n m	−	−	−	−	−	−	−	4 8 8 10	ジャンプ 各レジスタの内容番地へジャンプ 〔PC←HLなど〕 mn番地へジャンプ 〔PC←mn〕
JP NZ, mn JP Z, mn JP NC, mn JP C, mn JP PO, mn JP PE, mn JP P, mn JP M, mn	C2 n m CA n m D2 n m DA n m E2 n m EA n m F2 n m FA n m	−	−	−	−	−	−	−	10	ジャンプ(条件付) ・条件が成立すればmn番地へジャンプ (PC←mn) ・不成立なら本命令は無視する
JR e	18 e−2	−	−	−	−	−	−	−	12	ジャンプ・レラティブ(無条件) eバイト先へジャンプする 〔PC←PC+e〕
JR NZ, e JR Z, e JR NC, e JR C, e	20 e−2 28 e−2 30 e−2 38 e−2	−	−	−	−	−	−	−	条件成立 12 条件	ジャンプ・レラティブ(条件付) ・条件が成立すればeバイト先へジャンプ 〔PC←PC+e〕

ニーモニック	マシンコード	フラグ変化							所要クロックサイクル	動　作
		S	Z	H	P/V		N	C		
					P	V				
									不成立 7	・不成立なら本命令は無視する
LD A, n	3E n									8ビット転送（ロード）
LD A, A	7F									
LD A, B	78									A←（ソース）
LD A, C	79									
LD A, D	7A	—	—	—	—	—	—	—		
LD A, E	7B									
LD A, H	7C									
LD A, L	7D									
LD A, (mn)	3A n m								13	
LD A, (BC)	0A								7	
LD A, (DE)	1A								7	
LD A, (HL)	7E								7	
LD A, (IX+d)	DD 7E d								19	
LD A, (IY+d)	FD 7E d								19	
LD A, I	ED 57								9	8ビット転送　A←I A←R
LD A, R	ED 5F								9	IFF：0のとき割り込み禁止(DI) 1のとき割り込み可(EI)になっている
		・	・	0	IFF		0	—		LD　A, IとLD　A, Rではこの値がP/Vにコピーされる
LD B, n	06 n								7	8ビット転送（ロード）
LD B, A	47									B←（ソース）
LD B, B	40									
LD B, C	41									
LD B, D	42								4	
LD B, E	43	—	—	—	—	—	—	—		
LD B, H	44									
LD B, L	45									
LD B, (HL)	46								7	
LD B, (LX+d)	DD 46 d								19	
LD B, (IY+d)	FD 46 d								19	

ニーモニック	マシンコード	フラグ変化							所要クロックサイクル	動　作
		S	Z	H	P/V P	P/V V	N	C		
LD C, n	0E d								7	8ビット転送（ロード）
LD C, A	4F									
LD C, B	48									C←(ソース)
LD C, C	49									
LD C, D	4A								4	
LD C, E	4B	–	–	–	–	–	–	–		
LD C, H	4C									
LD C, L	4D									
LD C, (HL)	4E								7	
LD C, (IX+d)	DD 4E d								19	
LD C, (IY+d)	FD 4E d								19	
LD D, n	16								7	8ビット転送（ロード）
LD D, A	57									
LD D, B	50									D←(ソース)
LD D, C	51									
LD D, D	52								4	
LD D, E	53	–	–	–	–	–	–	–		
LD D, H	54									
LD D, L	55									
LD D, (HL)	56								7	
LD D, (IX+d)	DD 56 d								19	
LD D, (IY+d)	FD 56 d								19	
LD E, n	1E n								7	8ビット転送（ロード）
LD E, A	5F									
LD E, B	58									E←(ソース)
LD E, C	59								4	
LD E, D	5A									
LD E, E	5B	–	–	–	–	–	–	–		
LD E, H	5C									
LD E, L	5D									
LD E, (HL)	5E								7	
LD E, (IX+d)	DD 5E d								19	
LD E, (IY+d)	FD 5E d								19	
LD H, n	26 n								7	8ビット転送（ロード）
LD H, A	67								4	

ニーモニック	マシンコード	フラグ変化							所要クロックサイクル	動　作
		S	Z	H	P/V		N	C		
					P	V				
LD H, B	60									H←(ソース)
LD H, C	61									
LD H, D	62								4	
LD H, E	63	—	—	—	—	—	—	—		
LD H, H	64									
LD H, L	65									
LD H, (HL)	66								7	
LD H, (IX+d)	DD 66 d								19	
LD H, (IY+d)	FD 66 d								19	
LD L, n	2E								7	8ビット転送(ロード)
LD L, A	6F									
LD L, B	68									L←(ソース)
LD L, C	69									
LD L, D	6A								4	
LD L, E	6B	—	—	—	—	—	—	—		
LD L, H	6C									
LD L, L	6D									
LD L, (HL)	6E								7	
LD L, (IX+d)	DD 6E d								19	
LD L, (IY+d)	FD 6E d								19	
LD I, A	ED 47	—	—	—	—	—	—	—	9	8ビット転送 I←A
LD R, A	ED 4F								9	R←A
LD (mn), A	32 n m								13	8ビット転送(mn)←A
LD (BC), A	02	—	—	—	—	—	—	—	7	(BC)←A
LD (DE), A	12								7	(DE)←A
LD (HL), n	36 n								10	8ビット転送(ロード)
LD (HL), A	77									
LD (HL), B	70									(HL)←(ソース)
LD (HL), C	71	—	—	—	—	—	—	—		
LD (HL), D	72								7	
LD (HL), E	73									
LD (HL), H	74									
LD (HL), L	75									

ニーモニック	マシンコード	S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動作
LD (IX+d), n	DD 36 d n	—	—	—	—	—	—	—	19	8ビット転送（ロード） (IX+d) ←（ソース）
LD (IX+d), A	DD 77 d									
LD (IX+d), B	DD 70 d									
LD (IX+d), C	DD 71 d									
LD (IX+d), D	DD 72 d									
LD (IX+d), E	DD 73 d									
LD (IX+d), H	DD 74 d									
LD (IX+d), L	DD 75 d									
LD (IY+d), n	FD 36 d n	—	—	—	—	—	—	—	19	8ビット転送（ロード） (IY+d) ←（ソース）
LD (IY+d), A	FD 77 d									
LD (IY+d), B	FD 70 d									
LD (IY+d), C	FD 71 d									
LD (IY+d), D	FD 72 d									
LD (IY+d), E	FD 73 d									
LD (IY+d), H	FD 74 d									
LD (IY+d), L	FD 75 d									
LD BC, mn	01 n m	—	—	—	—	—	—	—	10	16ビット転送 BC←(ソース) mn　：定数 (mn)　：メモリの内容 メモリからレジスタの場合，たとえばHL, (mn)では L←(mn) H←(mn+1) となる
LD BC, (mn)	ED 4B n m								20	
LD DE, mn	11 n m	—	—	—	—	—	—	—	10	16ビット転送 DE←(ソース)
LD DE, (mn)	ED 5B n m								20	
LD HL, mn	21 n m	—	—	—	—	—	—	—	10	16ビット転送 HL←(ソース)
LD HL, (mn)	2A n m								16	
LD SP, mn	31 n m	—	—	—	—	—	—	—	10	16ビット転送 SP←（ソース）
LD SP, (mn)	ED 7B n m								20	
LD SP, HL	F9								6	
LD SP, IX	DD F9								10	
LD SP, IY	FD F9								10	
LD IX, mn	DD 21 n m	—	—	—	—	—	—	—	14	16ビット転送 IX←（ソース）
LD IX, (mn)	DD 2A n m								20	

ニーモニック	マシンコード	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V		N	C		
					P	V				
LD IY, mn LD IY, (mn)	FD 21 n m FD 2A n m	—	—	—	—	—	—	—	14 20	16ビット転送 IY←(ソース)

LD (mn), BC
LD (mn), DE
LD (mn), HL
LD (mn), SP
LD (mn), IX
LD (mn), IY
	ED 43 n m
	ED 53 n m
	22 n m
	ED 73 n m
	DD 22 n m
	FD 22 n m
		—	—	—	—	—	—	—	20
									20
									16
									20
									20
									20
										16ビット転送（ロード）
										(mn)←(ソース$_L$)
										(mn+1)←(ソース$_H$)

LDD	ED A8	×	×	0	—	·	0	—	16	ブロック転送 (ロード・デクリメント) (DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1
LDDR	ED B8	×	×	0	—	0	0	—	1バイトにつき21 最終のみ16	ブロック転送 (ロード・デクリメント・リピート) LDDをBC＝0までくり返す
LDI	ED A0	×	×	0	—	·	0	—	16	ブロック転送 (ロード・インクリメント) (DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1
LDIR	ED B0	×	×	0	—	0	0	—	1バイトにつき21 最終のみ16	ブロック転送 (ロード・インクリメント・リピート) LDIをBC＝0までくり返す
NEG	ED 44	·	·	·	—	·	1	·	8	ニゲイト 2の補数をとる A←0−A
NOP	00	—	—	—	—	—	—	—	4	何もしないで次へ (ノーオペレーション)

ニーモニック	マシンコード	S	Z	H	P/V		N	C	所要クロックサイクル	動作
					P	V				
OR n	F6 n								7	論理和(オア)
OR A	B7									A←A∨(ソース)
OR B	B0									a　b　a∨b
OR C	B1									0　0　0
OR D	B2								4	0　1　1
OR E	B3	·	·	0	·	–	0	0		1　0　1
OR H	B4									1　1　1
OR L	B5									
OR (HL)	B6								7	
OR (IX+d)	DD B6 d								19	
OR (IY+d)	FD B6 d								19	
OUT (C), A	ED 79									出力
OUT (C), B	ED 41									各レジスタの内容をBCレジスタの内容番地のポートへ出力
OUT (C), C	ED 49									
OUT (C), D	ED 51	–	–	–	–	–	–		12	〔I/O(BC)←(ソース)〕
OUT (C), E	ED 59									
OUT (C), H	ED 61									
OUT (C), L	ED 69									
OUT (n), A	D3 n	–	–	–	–	–	–		11	出力 I/O(An)←A
OUTD	ED AB	×	·	×	×	–	×	×	16	アウト・デクリメント B←B−1 I/O(BC)←(HL) HL←HL−1
OTDR	ED BB	×	1	×	×	–	×	×	1バイトにつき 21 最終のみ 16	アウト・デクリメント・リピート OUTDをB＝0までくり返す
OUTI	ED A3	×	·	×	×	–	×	×	16	アウト・インクリメント B←B−1 I/O(BC)←(HL) HL←HL+1

ニーモニック	マシンコード	フラグ変化 S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動作
OTIR	ED B3	×	1	×	×	–	×	×	1バイトにつき 21 最終のみ 16	アウト・インクリメント・リピート OUTIをB＝0までくり返す
POP AF	F1	–	–	–	–	–	–	–	10	16ビット転送（ポップ） スタックからレジスタへ転送 ディスティネーション$_L$←(SP) ディスティネーション$_H$←(SP＋1) SP←SP＋2
POP BC	C1									
POP DE	D1									
POP HL	E1									
POP IX	DD E1								14	
POP IY	FD E1								14	
PUSH AF	F5	–	–	–	–	–	–	–	11	16ビット転送(プッシュ) レジスタからスタックへ転送 (SP−1)←(ソース$_H$) (SP−2)←(ソース$_L$) SP←SP−2
PUSH BC	C5									
PUSH DE	D5									
PUSH HL	E5									
PUSH IX	DD E5								15	
PUSH IY	FD E5								15	
RES 0, A	CB 87	–	–	–	–	–	–	–	8	ビットリセット ソースの第0ビット←0
RES 0, B	CB 80									
RES 0, C	CB 81									
RES 0, D	CB 82									
RES 0, E	CB 83									
RES 0, H	CB 84									
RES 0, L	CB 85									
RES 0, (HL)	CB 86								15	
RES 0, (IX＋d)	DD CB d 86								23	
RES 0, (IY＋d)	FD CB d 86								23	
RES 1, A	CB 8F	–	–	–	–	–	–	–	8	ビットリセット ソースの第1ビット←0
RES 1, B	CB 88									
RES 1, C	CB 89									
RES 1, D	CB 8A									
RES 1, E	CB 8B									
RES 1, H	CB 8C									
RES 1, L	CB 8D									
RES 1, (HL)	CB 8E								15	

ニーモニック	マシンコード	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V		N	C		
					P	V				
RES 1, (IX+d)	DD CB d 8E								23	
RES 1, (IY+d)	FD CB d 8E								23	
RES 2, A	CB 97									ビットリセット
RES 2, B	CB 90									ソースの第2ビット←0
RES 2, C	CB 91									
RES 2, D	CB 92								8	
RES 2, E	CB 93	—	—	—	—	—	—	—		
RES 2, H	CB 94									
RES 2, L	CB 95									
RES 2, (HL)	CB 96								15	
RES 2, (IX+d)	DD CB d 96								23	
RES 2, (IY+d)	FD CB d 96								23	
RES 3, A	CB 9F									ビットリセット
RES 3, B	CB 98									ソースの第3ビット←0
RES 3, C	CB 99									
RES 3, D	CB 9A								8	
RES 3, E	CB 9B	—	—	—	—	—	—	—		
RES 3, H	CB 9C									
RES 3, L	CB 9D									
RES 3, (HL)	CB 9E								15	
RES 3, (IX+d)	DD CB d 9E								23	
RES 3, (IY+d)	FD CB d 9E								23	
RES 4, A	CB A7									ビットリセット
RES 4, B	CB A0									ソースの第4ビット←0
RES 4, C	CB A1									
RES 4, D	CB A2								8	
RES 4, E	CB A3	—	—	—	—	—	—	—		
RES 4, H	CB A4									
RES 4, L	CB A5									
RES 4, (HL)	CB A6								15	
RES 4, (IX+d)	DD CB d A6								23	
RES 4, (IY+d)	FD CB d A6								23	
RES 5, A	CB AF									ビットリセット
RES 5, B	CB A8									

ニーモニック	マシンコード	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V		N	C		
					P	V				
RES 5, C	CB A9									ソースの第5ビット←0
RES 5, D	CB AA									
RES 5, E	CB AB								8	
RES 5, H	CB AC	—	—	—	—	—	—	—		
RES 5, L	CB AD									
RES 5, (HL)	CB AE								15	
RES 5, (IX+d)	DD CB d AE								23	
RES 5, (IY+d)	FD CB d AE								23	
RES 6, A	CB B7									ビットリセット
RES 6, B	CB B0									ソースの第6ビット←0
RES 6, C	CB B1									
RES 6, D	CB B2								8	
RES 6, E	CB B3	—	—	—	—	—	—	—		
RES 6, H	CB B4									
RES 6, L	CB B5									
RES 6, (HL)	CB B6								15	
RES 6, (IX+d)	DD CB d B6								23	
RES 6, (IY+d)	FD CB d B6								23	
RES 7, A	CB BF									ビットリセット
RES 7, B	CB B8									ソースの第7ビット←0
RES 7, C	CB B9									
RES 7, D	CB BA								8	
RES 7, E	CB BB	—	—	—	—	—	—	—		
RES 7, H	CB BC									
RES 7, L	CB BD									
RES 7, (HL)	CB BE								15	
RES 7, (IX+d)	DD CB d BE								23	
RES 7, (IY+d)	FD CB d BE								23	
RET	C9	—	—	—	—	—	—	—	10	サブルーチンからのリターン PCへスタックよりPOP
RET NZ	C0								条件成立	条件付リターン
RET Z	C8								11	・条件が成立すれば POP PC
RET NC	D0									

ニーモニック	マシンコード	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V		N	C		
					P	V				
RET C	D8	—	—	—	—	—	—	—	不成立 5	不成立なら本命令は無視する
RET PO	E0									
RET PE	E8									
RET P	F0									
RET M	F8									
RETI	ED 4D	—	—	—	—	—	—	—	14	割り込み処理からのリターン POP PC
RETN	ED 45	—	—	—	—	—	—	—	14	ノンマスカブル割り込み処理からのリターン POP PC
RLA	17	—	—	0	—	—	0	·	4	ローテイト・レフト・アキュムレータ RL Aと同じ
RLCA	07	—	—	0	—	—	0	·	4	ローテイト・レフト・サーキュラ・アキュムレータ RLC Aと同じ
RRA	1F	—	—	0	—	—	0	·	4	ローテイト・ライト・アキュムレータ RR Aと同じ
RRCA	0F	—	—	0	—	—	0	·	4	ローテイト・ライト・サーキュラ・アキュムレータ RRC Aと同じ
RL A	CB 17	·	·	0	·	—	0	·	8	ローテイト・レフト Cy ← 7 ← 0 ← (Cy)
RL B	CB 10									
RL C	CB 11									
RL D	CB 12									
RL E	CB 13									
RL H	CB 14									
RL L	CB 15									
RL (HL)	CB 16								15	
RL (IX+d)	DD CB d 16								23	
RL (IY+d)	FF CB d 16								23	

ニーモニック	マシンコード	S	Z	H	P/V (P)	P/V (V)	N	C	所要クロックサイクル	動　作
RLC A	CB 07	·	·	0	·	—	0	·	8	ローテイト・レフト・サーキュラ（Cy ← 7 ← 0、ビット7 → ビット0）
RLC B	CB 00								8	
RLC C	CB 01								8	
RLC D	CB 02								8	
RLC E	CB 03								8	
RLC H	CB 04								8	
RLC L	CB 05								8	
RLC (HL)	CB 06								15	
RLC (IX+d)	DD CB d 06								23	
RLC (IY+d)	FD CB d 06								23	
RR A	CB 1F	·	·	0	·	—	0	·	8	ローテイト・ライト（Cy → 7 → 0、ビット0 → Cy）
RR B	CB 18								8	
RR C	CB 19								8	
RR D	CB 1A								8	
RR E	CB 1B								8	
RR H	CB 1C								8	
RR L	CB 1D								8	
RR (HL)	CB 1E								15	
RR (IX+d)	DD CB d 1E								23	
RR (IY+d)	FD CB d 1E								23	
RRC A	CB 0F	·	·	0	·	—	0	·	8	ローテイト・ライト・サーキュラ（7 → 0、ビット0 → ビット7 および Cy）
RRC B	CB 08								8	
RRC C	CB 09								8	
RRC D	CB 0A								8	
RRC E	CB 0B								8	
RRC H	CB 0C								8	
RRC L	CB 0D								8	
RRC (HL)	CB 0E								15	
RRC (IX+d)	DD CB d 0E								23	
RRC (IY+d)	FD CB d 0E								23	
RLD	ED 6F	·	·	0	·	—	0	—	18	ローテイト・レフト・ディジット（A: 7 4 | 3 0、(HL): 7 4 | 3 0）

ニーモニック	マシンコード	S	Z	H	P/V (P)	P/V (V)	N	C	所要クロックサイクル	動作
RRD	ED 67	・	・	0	・	—	0	—	18	ローテイト・ライト・ディジット A (HL) 7 4 3 0 7 4 3 0
RST 00H	C7	—	—	—	—	—	—	—	11	リスタート 0000 H ～0038 H のいずれかの番地に対する CALL
RST 08H	CF									
RST 10H	D7									
RST 18H	DF									
RST 20H	E7									
RST 28H	EF									
RST 30H	F7									
RST 38H	FF									
SBC A, n	DE n	・	・	・	—	・	1	・	7	8ビット引き算（キャリー付）（サブトラクト・ウィズ・キャリー） A←A—(ソース)—Cy
SBC A, A	9F								4	
SBC A, B	98								4	
SBC A, C	99								4	
SBC A, D	9A								4	
SBC A, E	9B								4	
SBC A, H	9C								4	
SBC A, L	9D								4	
SBC A, (HL)	9E								7	
SBC A, (IX+d)	DD 9E d								19	
SBC A, (IY+d)	FD 9E d								19	
SBC HL, BC	ED 42	・	・	×	—	・	1	・	15	16ビット引き算（キャリー付)(サブトラクト・ウィズ・キャリー） HL←HL—(ソース)—Cy
SBC HL, DE	ED 52									
SBC HL, HL	ED 62									
SBC HL, SP	ED 72									
SCF	37	—	—	0	—	—	0	1	4	セット・キャリーフラグ Cy←1
SET 0, A	CB C7									ビットセット
SET 0, B	CB C0									
SET 0, C	CB C1									

ニーモニック	マシンコード	フラグ変化							所要クロックサイクル	動　作
		S	Z	H	P/V		N	C		
					P	V				
SET 0, D	CB C2								8	ソースの第0ビット ←1
SET 0, E	CB C3	—	—	—	—	—	—	—		
SET 0, H	CB C4									
SET 0, L	CB C5									
SET 0, (HL)	CB C6								15	
SET 0, (IX+d)	DD CB d C6								23	
SET 0, (IY+d)	FD CB d C6								23	
SET 1, A	CB CF									ビットセット
SET 1, B	CB C8									
SET 1, C	CB C9									ソースの第1ビット
SET 1, D	CB CA								8	←1
SET 1, E	CB CB	—	—	—	—	—	—	—		
SET 1, H	CB CC									
SET 1, L	CB CD									
SET 1, (HL)	CB CE								15	
SET 1, (IX+d)	DD CB d CE								23	
SET 1, (IY+d)	FD CB d CE								23	
SET 2, A	CB D7									ビットセット
SET 2, B	CB D0									
SET 2, C	CB D1									ソースの第2ビット
SET 2, D	CB D2								8	←1
SET 2, E	CB D3	—	—	—	—	—	—	—		
SET 2, H	CB D4									
SET 2, L	CB D5									
SET 2, (HL)	CB D6								15	
SET 2, (IX+d)	DD CB d D6								23	
SET 2, (IY+d)	FD CB d D6								23	
SET 3, A	CB DF									ビットセット
SET 3, B	CB D8									
SET 3, C	CB D9									ソースの第3ビット
SET 3, D	CB DA								8	←1
SET 3, E	CB DB	—	—	—	—	—	—	—		
SET 3, H	CB DC									
SET 3, L	CB DD									
SET 3, (HL)	CB DE								15	

ニーモニック	マシンコード	フラグ変化							所要クロックサイクル	動作
		S	Z	H	P/V P	P/V V	N	C		
SET 3, (IX+d)	DD CB d DE								23	
SET 3, (IY+d)	FD CB d DE								23	
SET 4, A	CB E7									ビットセット
SET 4, B	CB E0									
SET 4, C	CB E1									ソースの第4ビット
SET 4, D	CB E2								8	←1
SET 4, E	CB E3	—	—	—	—	—	—	—		
SET 4, H	CB E4									
SET 4, L	CB E5									
SET 4, (HL)	CB E6								15	
SET 4, (IX+d)	DD CB d E6								23	
SET 4, (IY+d)	FD CB d E6								23	
SET 5, A	CB EF									ビットセット
SET 5, B	CB E8									
SET 5, C	CB E9									ソースの第5ビット
SET 5, D	CB EA								8	←1
SET 5, E	CB EB	—	—	—	—	—	—	—		
SET 5, H	CB EC									
SET 5, L	CB ED									
SET 5, (HL)	CB EE								15	
SET 5, (IX+d)	DD CB d EE								23	
SET 5, (IY+d)	FD CB d EE								23	
SET 6, A	CB F7									ビットセット
SET 6, B	CB F0									
SET 6, C	CB F1									ソースの第6ビット
SET 6, D	CB F2								8	←1
SET 6, E	CB F3	—	—	—	—	—	—	—		
SET 6, H	CB F4									
SET 6, L	CB F5									
SET 6, (HL)	CB F6								15	
SET 6, (IX+d)	DD CB d F6								23	
SET 6, (IY+d)	FD CB d F6								23	
SET 7, A	CB FF									ビットセット
SET 7, B	CB F8									

ニーモニック	マシンコード	フラグ変化							所要クロックサイクル	動　作
		S	Z	H	P/V		N	C		
					P	V				
SET 7, C	CB F9									ソースの第7ビット←1
SET 7, D	CB FA								8	
SET 7, E	CB FB	—	—	—	—	—	—	—		
SET 7, H	CB FC									
SET 7, L	CB FD									
SET 7, (HL)	CB FE								15	
SET 7, (IX+d)	DD CB d FE								23	
SET 7, (IY+d)	FD CB d FE								23	
SLA A	CB 27									シフト・レフト・アリスメティック
SLA B	CB 20									
SLA C	CB 21									Cy ← 7←0 ← 0
SLA D	CB 22								8	
SLA E	CB 23	・	・	0	・	—	0	・		
SLA H	CB 24									
SLA L	CB 25									
SLA (HL)	CB 26								15	
SLA (IX+d)	DD CB d 26								23	
SLA (IY+d)	FD CB d 26								23	
SRA A	CB 2F									シフト・ライト・アリスメティック
SRA B	CB 28									
SRA C	CB 29									
SRA D	CB 2A								8	Cy ← 7→0
SRA E	CB 2B	・	・	0	・	—	0	・		
SRA H	CB 2C									
SRA L	CB 2D									
SRA (HL)	CB 2E								15	
SRA (IX+d)	DD CB d 2E								23	
SRA (IY+d)	FD CB d 2E								23	
SRL A	CB 3F									シフト・ライト・ロジカル
SRL B	CB 38									
SRL C	CB 39									
SRL D	CB 3A								8	0 → 7→0 → Cy
SRL E	CB 3B	・	・	0	・	—	0	・		
SRL H	CB 3C									
SRL L	CB 3D									

ニーモニック	マシンコード	S	Z	H	P/V P	P/V V	N	C	所要クロックサイクル	動作
SRL (HL)	CB 3E								15	
SRL (IX+d)	DD CB d 3E								23	
SRL (IY+d)	FD CB d 3E								23	
SUB n	D6 n	·	·	·	—	·	1	·	7	8ビット引き算（サブトラクト）A←A−(ソース)
SUB A	97								4	
SUB B	90								4	
SUB C	91								4	
SUB D	92								4	
SUB E	93								4	
SUB H	94								4	
SUB L	95								4	
SUB (HL)	96								7	
SUB (IX+d)	DD 96 d								19	
SUB (IY+d)	FD 96 d								19	
XOR n	EE n	·	·	0	·	—	0	0	7	排他的論理和（エクスクルーシブ・オア）A←A⊕(ソース)
XOR A	AF								4	
XOR B	A8								4	
XOR C	A9								4	
XOR D	AA								4	
XOR E	AB								4	
XOR H	AC								4	
XOR L	AD								4	
XOR (HL)	AE								7	
XOR (IX+d)	DD AE d								19	
XOR (IY+d)	FD AE d								19	

a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0

付録3．1バイト符号つき16進数

(欄外は，16進数
欄内は，10進数)

上位＼下位	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	-128	-127	-126	-125	-124	-123	-122	-121	-120	-119	-118	-117	-116	-115	-114	-113
9	-112	-111	-110	-109	-108	-107	-106	-105	-104	-103	-102	-101	-100	-99	-98	-97
A	-96	-95	-94	-93	-92	-91	-90	-89	-88	-87	-86	-85	-84	-83	-82	-81
B	-80	-79	-78	-77	-76	-75	-74	-73	-72	-71	-70	-69	-68	-67	-66	-65
C	-64	-63	-62	-61	-60	-59	-58	-57	-56	-55	-54	-53	-52	-51	-50	-49
D	-48	-47	-46	-45	-44	-43	-42	-41	-40	-39	-38	-37	-36	-35	-34	-33
E	-32	-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17
F	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

索　　引

≪欧　文≫

≪和　文≫

【ア】

【イ】

【オ】

【カ】

【キ】

【ク】

【コ】

【サ】

【シ】

【ス】

【セ】

【ソ】

【タ】

【テ】

【ニ】